Inaugural-Dissertation

zur Erlangung der Doktorwürde der

Gesamtfakultät für Mathematik,
Ingenieur- und Naturwissenschaften

der

Ruprecht-Karls-Universität
Heidelberg

vorgelegt von

Satya (Shideh) Almasian, M.Sc.

aus Teheran

Tag der mündlichen Prüfung:

Quantity-centric Search and Retrieval

by

Shideh (Satya) Almasian

Supervisor: Prof. Dr. Michael Gertz
Second referee: Prof. Dr. Stefan Riezler

Zusammenfassung

Quantitäten sind unerlässlich, wenn es um die Darlegung von Fakten in Anwendungsbereichen wie z.B. der Finanzbranche, Volkswirtschaft, Medizin und allgemeinen Wissenschaft geht. Diese Dissertation beinhaltet 1.423 Quantitäten. Während dies gerade einmal 1% der gesamten Anzahl an Wörtern ausmacht, beschreiben diese Wertangaben die genauesten und wichtigsten Informationen, die für die Analyse und den Vergleich von Systemen notwendig sind. Trotz dieser Relevanz von Quantitäten gibt es nur wenige Studien, die sich mit der textuellen Repräsentation und deren Auswirkungen auf das Information Retrieval (IR) beschäftigen. In einer Vielzahl von Anwendungen spielen Quantitäten eine zentrale Rolle, um den Informationsbedarf von Nutzern zu decken, und sie können ohne semantisches Verständnis nicht richtig behandelt werden. Bei der Suchanfrage „ein Gebrauchtwagen mit weniger als 200 PS“ sucht ein Nutzer beispielsweise nach einem Auto in einem gewissen Bereich der Suchparameter. Um auf diese Suche eine richtige Antwort zu geben, muss ein Suchsystem nicht nur den Zusammenhang zwischen einem Auto und der zugehörigen Quantität verstehen, sondern auch die entsprechenden Werte und Einheiten verarbeiten. Darüber hinaus sollten nur Ergebnisse präsentiert werden, ienen der Wert für dieses spezifische Attribut kleiner als „200“ ist, was ein Verständnis von Nachbarschaft von Zahlen erfordert. Derzeitige Modelle zur Repräsentation von Quantitäten betrachten diese isoliert und vernachlässigen dabei den Zusammenhang zu benachbarten Token im Text. Weiterhin wenden moderne Suchmaschinen die gleichen Methoden auf Wörter und Quantitäten an und ignorieren dabei Informationen zu Werten und Einheit von Quantitäten. Somit führen quantitäts-zentrische Suchanfragen oft zu irrelevanten Ergebnissen und Nutzer verlieren wertvolle Zeit beim Anschauen durch irrelevante Inhalte.

Diese Arbeit beschäftigt sich mit diesen Problemen und zielt darauf ab, das Quantitäts-Verständnis von derzeitigen Suchsystemen zu verbessern. Wir beginnen mit einem gesamtheitlichen Modell zur Repräsentation von Quantitäten, welches effektiv Kombinationen von *Werten* bzw. *Einheiten*, *Veränderungen* im Verhalten einer Quantität im gegebenen Kontext (bespielsweise fallend oder steigend) und *Konzepten* (verwandte Entitäten oder Ereignisse) erkennen kann. Auf Basis dieses Modells wird eine Methode zur Extraktion namens Comprehensive Quantity Extraction (CQE) entwickelt. Darüber hinaus stellen wir einen neuartigen Benchmark-Datensatz vor, der speziell für die Bewertung dieser Aufgabe entwickelt wurde. Mithilfe dieser Extraktionsmethode stellen wir zwei quantitäts-fokussierte Suchmethoden vor, welche sowohl klassische als auch neuronale Modelle umfassen. Diese Modelle sind so konzipiert, dass sie sowohl die Nachbarschaft von Quantitäten als auch den Text in ein Ranking mit einbeziehen. Eine Variante ist der sogenannte Disjoint-Ranker, welcher die Relevanz von Quantitäts- und Texttoken mittels einer Quantitäts-Indexstruktur separat bewertet. Die zweite Variante, der Joint-Ranker, realisiert eine gemeinsame Abbildung von Quantitäten und Textinhalten mittels Fine-Tuning von neuronalen Netzen auf Daten, welche viele Quantitäten beinhalten. Diese Techniken beziehen Mengeninformationen während des Rankings sowohl in neuronale als auch in lexikalische Modelle ein, mit minimalem Overhead in Bezug auf die Effizienz und ohne Änderung der zugrundeliegenden Architektur. Diese Modelle können Suchanfra-

gen auswerten, die numerische Bedingungen wie *gleich*, *größer als* und *kleiner als* sowie bestimmte Wörter enthalten. Um die Effektivität unserer Ranking-Modelle zu bewerten, stellen wir zwei neue Benchmark-Datensätze aus dem Finanz- und Medizinbereich vor.. Wir vergleichen unsere Methoden in den Benchmarks mit verschiedenen klassischen und neuronalen Retrieval-Systemen und zeigen eine signifikante Verbesserung bei der Beantwortung quantitätfokussierter Abfragen.

Abstract

Quantities are essential in documents to describe factual information in domains such as finance, business, medicine, and science. This thesis alone encompasses 1,423 quantities within its text. While these account for just 1% of the overall word count, these values contain the most precise and crucial information necessary for analysis and system comparison. Despite the importance of quantities, only a handful of studies focus on their representation in text and their impact on Information Retrieval (IR). In many cases, the information needs of a user revolve around quantities and cannot be resolved without understanding their semantics. For instance, in the query "a used car that has less than 200hp", the user is looking for a car with a specific parameter range. To provide an accurate response, the retrieval method should not only recognize the connection between the car and the quantity in the query but it must also comprehend value comparisons and units. Furthermore, the retrieved results should contain values less than "200" for this specific attribute of a car, requiring an understanding of numerical proximity. However, current quantity models often analyze values and units in isolation, disregarding their relationships to other tokens in the text. Additionally, modern search engines apply the same ranking mechanisms to both words and quantities, overlooking magnitude and unit information. As a result, quantity-centric queries yield sub-par results and often cost the users valuable time navigating through irrelevant content.

In this thesis, we address these shortcomings and aim to enhance the quantity understanding of current IR models. We start by presenting a holistic quantity model that efficiently models combinations of *values* and *units*, *changes* in the behavior of a quantity in the given context (e.g., rising or falling), and the *concept* (related entities or events) of a quantity. This quantity model leads to the development of an extraction framework called Comprehensive Quantity Extraction (CQE), which is designed to detect and normalize quantities in text. Additionally, we introduce a novel benchmark dataset tailored to evaluate quantity extraction.

Using the quantity extractor, we introduce two quantity-aware retrieval techniques that encompass both classical and neural models. These models are designed to rank documents based on the proximity of quantities in the text as well as the textual content. One method is the disjoint quantity-aware ranker, which is designed to separate the ranking of quantities and textual tokens by means of a quantity index structure. The second method is the joint quantity-aware ranker, which focuses on the joint ranking of quantities and textual tokens by fine-tuning a neural retrieval model on quantity-rich data. These techniques incorporate quantity information during ranking in both neural and lexical models, with minimal overhead in terms of efficiency and without the change in the system. These models can answer queries containing the numerical conditions *equal*, *greater than*, and *less than* as well as keyword search. To evaluate the effectiveness of our ranking models, we introduce two novel benchmark datasets in the domains of finance and medicine. We compare our methods on the benchmarks against various classical and neural retrieval systems and show significant improvement in answering quantity-centric queries.

Acknowledgements

"Without great people, even great ideas are useless."

Simon Sinek

First and foremost, I owe my deepest gratitude to my supervisor Prof.Dr.Michael Gertz. Not only did he make it possible for me to pursue a PhD, but his support, patience, and open and honest attitude transformed what is for most people the most difficult time of their life, into one of the most happy and rewarding experiences of mine. He was truly the most excellent supervisor I could ever ask for. I would also like to express my gratitude to my second supervisor Prof.Dr. Stefan Riezler for constructive discussions and feedback in our meetings.

I am thankful for the support that I received from the Faculty of Mathematics and Computer Science at Heidelberg University, the Epinetz project, and the members of the Data Science group. Special thanks go to my colleague and partner in crime, Dennis Aumiller, for many hours of fruitful discussions, and collaboration and for believing in me even when I did not. Academic accomplishments aside, the most valuable gain during this time was his friendship.

I appreciate the assistance of Milena Bruseva on numerous data cleaning and dataset preparation tasks and her diligence in the face of my many requests. I would also like to thank Vivian Kazakova for her help in data annotation and implementation of the quantity extractor in this thesis as well as David Pohl and Alexander Kosnac for extremely enjoyable collaborations. My thanks also go to my predecessor, mentor, and good friend, Andreas Spitz for paving the way and motivating me to pursue a PhD in the first place. Furthermore, I wish to thank all my colleagues at the chair, Philip Hauser, Sebastian Lackner, John Ziegler, Jayson Salazar, Ashish Chouhan, and Nicolas Reuter. I am also thankful for the people who read this thesis completely or in parts. Great thanks are due to Richard Tapper, Philip Hauser, Ashish Chouhan, John Ziegler, Marina Walter, and Ghadeer Mobasher.

I am grateful to my colleagues at Amazon, especially Omar Zaidan, my manager for an exceptionally fruitful internship experience. It truly transformed how I go about research.

Many thanks go to those who worked tirelessly to keep everything running smoothly, including Natalia Ulrich and Catherine Proux-Wieland on the administrative level as well as Holger Wünsche, Milena Bruseva, and Saifeldin Mandour, our system administrators.

A tree is only as strong as its roots and my roots during this time were my family and friends. My

gratitude goes to Felix Feldmann, the unwavering anchor by my side, for his everlasting support, tolerance, and loyal presence. He was always ready to lend a helping hand and resolved many issues before they even reached my doorstep. I owe immense gratitude to my mother, Sima Mirhosseini, for her unfaltering support and patient ear amid my ceaseless life dramas. Many thanks to Ziba Mirhosseini and Richard Tapper, who encouraged me and made it possible for me to pursue an education in Germany.
Finally, I express my gratitude to many others who shall remain nameless. Their anonymity in no way diminishes the significant role they have played in shaping my life.

Thank you.

Contents

1 Introduction

> "Number rules the universe."
>
> *Pythagoras*

If you are on a search for a used car on a budget below €5k or trying to buy a small fridge that fits exactly in the designated corner of the kitchen, prepare for yourself hours of frustration. If your query contains quantities and numerical conditions, modern search engines are likely to misinterpret your request. Consider the screen-shot from the well-known e-commerce website of Amazon [1] in Figure 1.1 for the query "fridge with less than 88L". All the results on the first page (only three shown here for brevity) are either fridges that have a capacity of exactly 88L (first example) or significantly larger than the specified amount. This is because conventional search methods struggle to grasp the semantics of numerical conditions, values, and scales and fail to retrieve the desired results.

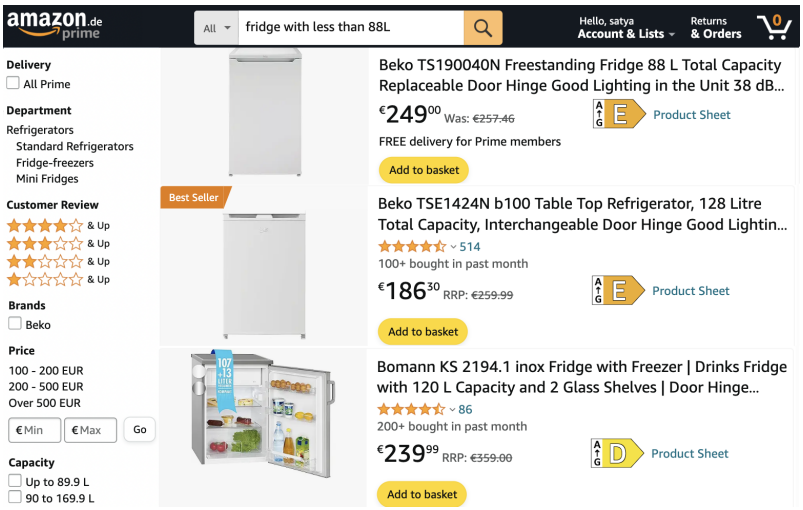


Figure 1.1: Amazon product search for the query "fridge with less than 88L"; Taken on 11.04.2024. The image is edited to make the demonstartion of results more compact.

While one option to narrow down the search space is to utilize the faceted search available in the left panel, this navbar is populated with information from a structured database, where specific values for product attributes should be entered manually. If you have encountered a form with numerous

[1] https://www.amazon.de/ (last accessed 16.04.2024)

optional fields, you are likely familiar with how few users actually complete all the information. In reality, users often resort to either cramming all product details into a lengthy title or burying them in verbose product descriptions that current search models struggle to comprehend and analyze effectively. Moreover, narrowing the search space would have not been necessary if the user intent had been correctly identified from the query itself.

Incorrect interpretation of numerical conditions is not the only issue. The absence of a relationship between the quantity values and their units complicates matters further. If the value "88" appears anywhere in the product description, the model is prone to detecting it, irrespective of the correct unit. For instance, the height of "88cm" can be confused with capacity of "88L". In such scenarios, users must invest significant time scrolling through product pages to find the desired match.

The importance of quantities is not limited to product search. In domains such as finance, medicine, and even legal documents, searching for specific values becomes important. Consider a financial analyst going through a multitude of earning call transcripts and stock market reports per day. Each document contains numerous quantities, tables, and graphs. Manually searching through these documents to find specific information is a time-consuming and tedious task. Faced with a corpus rich in quantitative data, current *Information Retrieval (IR)* systems reliant on semantic or lexical similarity are ill-equipped to answer quantity-centric queries.

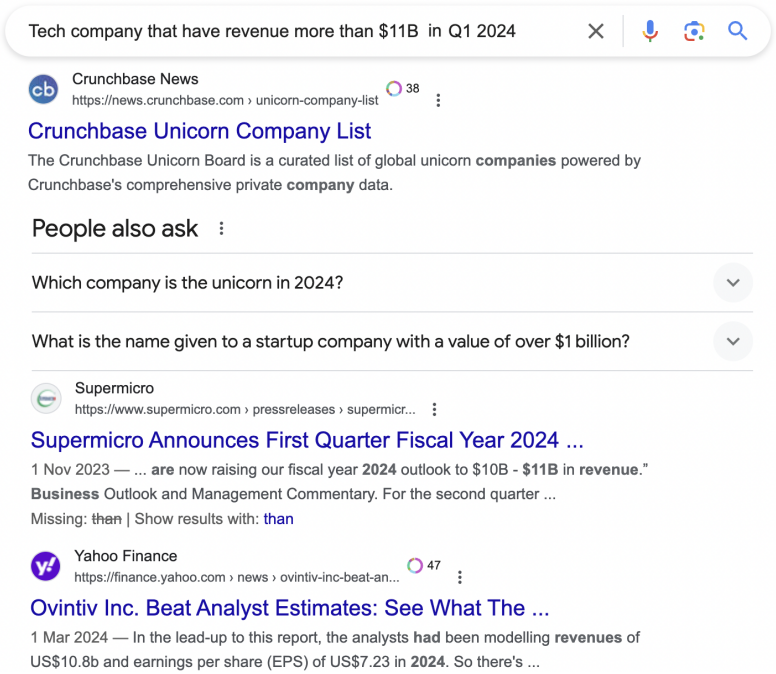


Figure 1.2: Google results for the query "Tech company that have revenue more than $11B in Q1 2024"; Taken on 11.04.2024. The questions *people also ask* are cut down to two, where the other two were irrelevant questions: "What are the top unicorn companies in the US?", "How many unicorn companies are there in China?", none of them reflecting the user's intention.

For instance, a financial analysis might look for "Tech company that has revenue more than $11B in Q1 2024". If one tries the same query on Google, [2], which is currently the most used search engine in the world, the result demonstrated in Figure 1.2 would pop up. None of the top results reflect the user's information needs. Even the questions generated for the *people also ask* section contain the wrong value, i.e., "$1 billion" vs "$11 billion".
Considering the two examples, we notice that search engines tend to prioritize lexical matches for quantities (as seen in the second result in Figure 1.2 and the first result in Figure 1.1), and disregard the numerical condition (the word "than", which is part of the numerical condition, was even omitted from the query text in Figure 1.2). Consequently, the ranking process overlooks a crucial aspect of the user's query: the quantity information, leading to suboptimal results.

1.1 Motivation

How is it that sophisticated search architectures, so adapt at reading users' intentions from a few keywords, fail when it comes to quantitative information? Considering how ubiquitous quantities are and how crucial they are in certain domains, is it not strange how the modern search architectures are inattentive to numbers and often neglect them altogether?
The main reason for the strange behavior of these search systems is that retrieval models treat quantities as second-class citizens. The problem begins with neglecting quantities in tokenization and persists through learning representation for natural language, all the way to training objectives and definition of relevance for IR systems.
The most prominent methods for tokenization (word or sub-word level) split sentences on punctuation and whitespace. In the case of the popular Byte pair encoding, a single number is often split into distinct tokens, where, unlike words, these splits are not meaningful. As an example, by using Byte pair encoding (Gage, 1994), the value "1,000,000" would be segmented into three tokens: "1,", "000," and "000", as segmentation occurs based on punctuation. Additionally, even a value without punctuation, such as "12300" might be split into "123' and "00", based on the frequency of distinct sub-words in the training corpus of the tokenizer.
What is even more troublesome is that during representation learning these discretized units are used to represent a continuous entity. Consequently, the scale and relationships between values are not preserved, and information about units is disregarded. Unlike words, whose semantics can be derived based on the context they occur in (Firth, 1957), the same context can result in vastly different ranges of values. Quantity proximity is pre-defined on an interval scale and not by surrounding words. Take the earning reports of a company into account. Although the context and the vocabulary used in the report remain relatively the same throughout the years, the values reported change based on the current state of the company. Therefore, masked language modeling (Devlin et al., 2019a) or other denoising objectives (Raffel et al., 2020), which are often used, are not well-suited

[2] https://www.google.com/ (last accessed: 06.05.2024)

to capture numeracy. Since numbers usually make up a smaller portion of the corpus, most researchers choose to ignore them or treat them the same way as other textual tokens.
These inadequately equipped language models and embeddings serve as the foundation for modern IR systems, where at the retrieval level, quantities and their semantics are once again overlooked. IR systems are trained based on topical similarity and the notation of relevance defined in these systems does not consider quantity comparisons. This phenomenon cannot solely be attributed to the model itself but also arises from the datasets used for optimization. The majority of training and test data for IR comes from a set of general-domain benchmarks (Nguyen et al., 2016) that are frequently utilized and overly optimized by researchers in the field. These datasets contain mainly term-based queries and as a result, focusing on quantities has little trade-off for researchers. However, domain-specific and real-world scenarios usually differ from general domain benchmark data, and queries involving quantities occur more frequently.
In this thesis, we focus on exploring solutions for bringing quantity awareness into current IR systems, thereby alleviating some of these issues, while making minimal alterations to the original retrieval objectives. The goal is not to develop a specialized system solely for quantity-aware retrieval, but an IR system that can retain its textual ranking capabilities but also extends to ranking quantities and considering numerical conditions. This system should understand numerical proximity and capture relationships between values and units. At the same time, it should meet the criteria for an effective and applicable IR system. This entails being efficient in retrieving results, readily adaptable to new domains, and performing effectively in low-resource settings.
Quantity-centric queries can be defined in a variety of ways. In this thesis, however, we focus on a special type of query, where a numerical condition is imposed on a quantity. We found such queries to be most problematic for the current search architectures, as demonstrated by the examples mentioned above. Quantity-centric queries contain a search term, the same as any other query type, but there is a condition imposed on an attribute of the item, entity, or event that the user is searching for. An example of such query is “fridge below 114 V”, where the user is seeking a very specific fridge. The voltage is implicitly expressed through the unit “V” and the numerical condition indicates that it should not exceed “114”.
In the following, we summarize our contributions towards creating a quantity-aware retrieval framework that can efficiently answer quantity-centric queries, and give an overview of the thesis structure.

1.2 Contribution and Structure

As we have stated above, the poor performance of IR systems is not due to a single reason but it is attributed to many parts, many of which extend beyond the retrieval system itself. Although exploring all these parts is beyond the scope of a single thesis, we identify core elements for addressing these issues. We start by introducing a quantity model that takes the contextualized information into account. Using this contextualized model of quantities, we explore two approaches to bring

quantity awareness into current retrieval architectures. Since our contributions are aligned with the structure of the thesis, we present them jointly, in the following.

I To find the viable strategy for building a quantity-aware retrieval system, we first need to understand the current retrieval architectures and how they represent quantities. In Chapter 2, we provide an in-depth overview of the commonly used retrieval models, including the probabilistic model of BM25 (Robertson and Zaragoza, 2009) and transformer-based retrieval systems (Zhao et al., 2022). The inner workings of these models are important as they make the basis for our quantity-aware retrieval systems.

II As mentioned before, there has been limited work directly related to IR and quantities. In Chapter 3, we discuss these few works and we set out to investigate and gain inspiration from other domains. One relevant area that we also explore in Chapter 4 is describing a quantity model and a method for quantity extraction. Other related work includes the investigation of numeracy in language models and word embeddings and dedicated question-answering models for numerical reasoning. Efforts in these directions provide grounds for bringing quantity understanding to IR methods as well.

III Based on these building blocks, we proceed to the definition of a suitable quantity model. We formally introduce a contextualized quantity model in Chapter 4 that extends the quantity models in previous work to include context information. Unlike previous approaches, we avoid studying quantities as value and unit pairs in isolation but aim to capture the relationship with textual content. Alongside the quantity model, we introduce a *Comprehensive Quantity Extractor (CQE)* that can extract these contextualized quantities from the text. Moreover, we propose a novel benchmark dataset for quantity extraction and perform extensive evaluation against other extractors.

IV Having introduced a suitable quantity model, we explore various ways to bring quantity understanding into IR models. In Chapter 5, we propose two novel methods designed to rank both quantity and textual content either jointly or independently. The disjoint quantity-aware ranker is inspired by the fact that quantities are inherently different from other textual tokens and their proximity should be computed separately. On the other hand, the joint approach focuses on enhancing quantity understanding in neural retrieval models by additional task-specific fine-tuning. To evaluate the effectiveness of our proposed models, we introduce two novel quantity-centric benchmark datasets in the domains of finance and medicine and compare our method against various lexical and neural models.

Finally, in Chapter 6, we summarize the contributions that we have made, the limitations of the proposed models, and discuss future research directions. Our contributions are based on a set of peer-reviewed publications:

I Satya Almasian, Milena Bruseva, and Michael Gertz. QFinder: A Framework for Quantity-centric Ranking. In *SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022*, pages 3272–3277. ACM, 2022b. URL https://doi.org/10.1145/3477495.3531672

II Satya Almasian, Vivian Kazakova, Philip Göldner, and Michael Gertz. CQE: A Comprehensive Quantity Extractor. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 12845–12859. Association for Computational Linguistics, 2023. URL https://aclanthology.org/2023.emnlp-main.793

III Satya Almasian, Alexander Kosnac, and Michael Gertz. QuantPlorer: Exploration of Quantities in Text. In *Advances in Information Retrieval - 46th European Conference on Information Retrieval, ECIR 2024, Glasgow, UK, March 24-28, 2024, Proceedings, Part V*, volume 14612 of *Lecture Notes in Computer Science*, pages 171–176. Springer, 2024b. URL https://doi.org/10.1007/978-3-031-56069-9_13

IV Satya Almasian, Vivian Kazakova, and Michael Gertz. Numbers Matter! Bringing Quantity-awareness to Retrieval Systems. Under Review, 2024a

2 Background

> "Know where to find the information and how to use it – That's the secret of success."
>
> *Albert Einstein*

Information Retrieval (IR) deals with the representation, storage, organization, and access of information items (Baeza-Yates and Ribeiro-Neto, 2011). Retrieval is performed on a variety of information items, from text to images and video. In this thesis, however, we focus solely on textual content. In this chapter, we cover the foundations and background of textual retrieval systems, where the information items are snippets of text. The granularity of a text snippet depends on the purpose of retrieval and varies from sentences to paragraphs or an entire document. Generally, given a search query x and a corpus of documents $D = \{d_1, d_2, ..., d_n\}$, a retrieval system ranks the documents in D based on their relevance to the query. In other words, it estimates a probability distribution over the corpus of documents $p(d_i|x)$ of how probable document d_i is, given x. [1]

The role of retrieval systems is to estimate the relevance of a query (x) to a document (d_i) based on the similarity of their representations. If τ is a function that extracts a representation from raw text inputs and sim is a similarity function that calculates the similarity between $\tau(x)$ and $\tau(d_i)$, then the probability $p(d_i|x)$ is estimated as shown in Equation 2.1.

$$p(d_i|x) = \frac{exp(sim(\tau(x), \tau(d_i)))}{\sum_{j=1}^{|D|} exp(sim(\tau(x), \tau(d_j)))} \tag{2.1}$$

Basically, Equation 2.1 calculates the similarity of query and document representations normalized over the entire collection, resulting in the probability of their relevance. This equation can be used to describe all retrieval approaches, regardless of their complexity. Therefore, the challenge of creating a retrieval system boils down to defining a suitable representation function (τ), with respect to a similarity function (sim). The similarity function is often not the bottleneck, as simple functions like cosine similarity or dot product have proven to be very effective.
It is worth noting, that although here we calculate the similarity of query and document representations, the notion of similarity is not the same as the relevance. Similarity refers to the degree of like-

[1] In the glossary, notations used throughout this thesis are provided, mainly adapted from (Mitra and Craswell, 2018).

ness or resemblance, whereas relevance refers to the usefulness of a document in satisfying a user's information need or answering a specific query. Relevance is about how well a document meets the user's expectations and goes beyond mere similarity. Relevance considers the quality, context, and intent behind the query and the document. As a result, a document might be highly relevant, even if it does not share many common terms with the query. For instance, a document explaining "how to bake a cake" is relevant to the query "cake baking instructions" even if they do not have many identical terms. However, in IR when we refer to similarity calculation, relevance is intended.

When studying IR in academia, the document collection often remains static, and new and diverse queries are submitted to the system. This form of retrieval is known as *ad-hoc* retrieval, and this thesis focuses on this task. However, in real-world scenarios, the document collection is rarely static and new documents are added continuously to the collection. On the other hand, *filtering* refers to the case where the queries are static and new documents are added continuously to the corpus (Baeza-Yates and Ribeiro-Neto, 2011).
Baeza-Yates and Ribeiro-Neto (2011) divide classical IR models based on the representations of query and document into *Boolean*, *probabilistic*, and *vector* models. In Boolean models, documents and queries are represented as a set of terms, usually stored in an index structure. Probabilistic models use probability theory to compute the relevance between documents and queries, and in vector models, queries and documents are represented as vectors in some high dimensional space. The earliest approaches to IR are Boolean, with vector and probabilistic models appearing later in the literature. Traditional models in these categories are *lexical* or word-based, focusing on the presence or absence of terms. Although decades old, these models are still very powerful baseline systems. With the emergence of embedding techniques, vector-based models transformed into dense representations capturing the *semantics* of documents, driven by the meaning of words. The latest approaches employ deep neural networks to learn such representations that capture more intricate relationships between tokens in the text. In the upcoming sections, we give an overview of the important classical IR methods, followed by neural models and semantic-focused representations.

2.1 Classical Information Retrieval

In this section, we discuss the most prominent classical methods in IR. Insights from these models not only build the foundation of more recent approaches but also serve as powerful baselines. Classical models mostly work on the *term* or *word* level in a document, where a term or a word is the most atomic unit of meaning in language. Later, neural models look at more fine-grained units in text, namely *sub-words*, which are commonly occurring sequences of characters in text. As a result, *tokens* encompass both words and sub-words, depending on the chosen tokenization method. Throughout this thesis, the term *token* refers to either word or sub-word, in accordance with the assumptions of the model being discussed.

Studying classical IR models becomes important in Chapter 5, where we discuss quantity-aware rankers. One approach we introduce for quantity-aware IR leverages inverted indices, tracing its origins back to Boolean and vector-based models. Additionally, our lexical quantity-ranker is grounded in probabilistic models, specifically BM25 (Robertson and Jones, 1976), which we discuss in detail in this chapter.

2.1.1 Boolean Models

The Boolean retrieval model is a simple retrieval system based on set theory and Boolean algebra, such that the concept behind it is easy to grasp by a common user. The Boolean retrieval model represents any query as a Boolean expression of terms, in which terms are combined with the operators AND, OR, and NOT (Roelleke et al., 2009). In these models, documents are represented as a set of index terms. Index terms are either absent or present in a document, i.e., their weights are binary. The information about the corpus is stored in an index structure with *posting lists*, where a posting list is a record of all documents the term appears in. The entire structure for all documents in the corpus is referred to as an *inverted index* (Manning et al., 2008). Such an index allows for efficient finding of relevant documents in a large corpus. In Chapter 5, we use a similar structure in the disjoint quantity-ranker model, to represent quantities in the corpus and compute relevance based on quantity proximity. To create an inverted index:

1. Tokenize the documents in the collection. Tokenization is the task of splitting a document into tokens. In the case of Boolean models, usually word-level tokenization is used.

2. Apply linguistic preprocessing for normalizing the text, such as lemmatization or stemming. Stemming and lemmatization are ways to reduce the inflectional forms of a word to a common base form. Stemming removes the ends of words, whereas lemmatization removes inflectional endings and returns the base form of a word, called a lemma (Manning et al., 2008).

3. Identify the index terms, which are important terms for retrieval. Often stopwords and other unnecessary tokens like punctuation are removed at this stage and other tokens are kept for indexing. The set of all the index terms from the vocabulary.

4. Create an inverted index from the index terms to posting lists of document ids.

An example of an inverted index is shown in Figure 2.1, for three example terms from a vocabulary. The Boolean model uses exact matches between a user's query and the documents that satisfy the Boolean condition. An example query to match the index on Figure 2.1 is "Dog AND Cat". To find the relevant documents, first "Dog" has to be located in the dictionary and the respective posting list has to be retrieved. Then, the same is repeated for "Cat". The intersection of the two posting lists retrieves the final documents, i.e., [2, 17, 22]. There are no partial matches in this model, and the generated response is always binary: the document is relevant (1) or is not (0) relevant. This

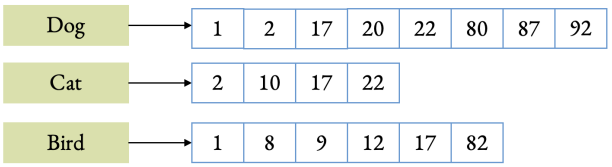


Figure 2.1: An example of an inverted index. The dictionary is commonly kept in memory for efficient access, which points to posting lists stored on disk.

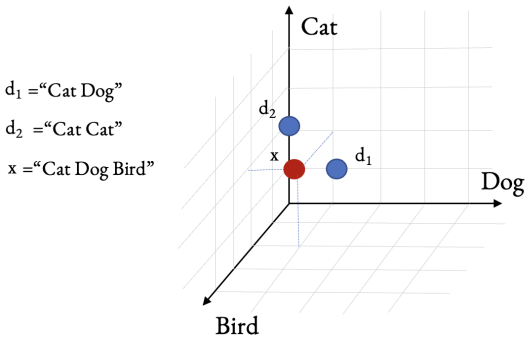


Figure 2.2: Two documents and a query with a very limited vocabulary in the vector space model.

is one of the major drawbacks of Boolean models, as there is no in-between grading present. Moreover, Boolean expressions have very precise semantics, and it is not simple to translate all information needs into Boolean expressions. These disadvantages result in retrieval of too few or too many documents (Baeza-Yates and Ribeiro-Neto, 2011).

2.1.2 Vector Space Models

Vector space models (Salton, 1975) overcome the limitations of Boolean models with an algebraic solution that can perform partial matches. In Boolean models, the relevance between documents and queries is measured by the appearance of shared index terms. The idea carries over to vector space models, with the difference that the binary weighting is changed to a frequency-based weighting schema. Computing relevance based on term frequencies and the frequency-based weighting schema is an important prerequisite in understanding of the BM25 model. In this section, we discuss the important concepts in this regard.

In vector space models, queries and documents are represented by a vector in n-dimensional space (Berry et al., 1999). The dimensions, or the basis of vectors, are defined by the number of terms in the vocabulary and are linearly independent. This is a major drawback of these systems, as in the real world typically terms are not semantically independent, e.g., "river" and "boat" are semantically similar and they occur in the same context. If we limit the terms in the vocabulary to "Cat", "Dog", and "Bird', a sample representation of two documents and a query is shown in Figure 2.2.

The similarity between a query and the document is measured by their distance in space. The standard way of quantifying the similarity is with *cosine similarity* (as shown in Equation 2.2), where

the representations of a document and a query are denoted as $\tau(x) = \vec{x} = (\omega_{1,x}, \omega_{2,x}, ..., \omega_{t,x})$ and $\tau(d) = \vec{d} = (\omega_{1,d}, \omega_{2,d}, ..., \omega_{t,d})$. ω is the weight in each dimension, indicating the frequency of the term (Manning et al., 2008). The denominator is a length-normalizer and, in the case of unit vectors, can be written as the dot-product $sim(d, x) = \vec{x} \cdot \vec{d}$. The vectors are usually sparse, since each document uses a limited set of words from the vocabulary. Hence, the dot product is generally inexpensive to compute.

$$sim(d, x) := \frac{\vec{x} \cdot \vec{d}}{||\vec{x}|| \cdot ||\vec{d}||} = \frac{\Sigma_{i=1}^{|x|} \omega_{i,d} \cdot \omega_{i,x}}{\sqrt{\Sigma_{i=1}^{|x|} \omega_{i,d}^2} \sqrt{\Sigma_{i=1}^{|x|} \omega_{i,x}^2}} \tag{2.2}$$

Unlike Boolean models, similarity is not limited to binary relevance as the cosine similarity returns a *degree of similarity* between a document and a query. The most relevant documents are obtained by establishing a threshold on $sim(d, x)$.
The variety between vector space models comes from different representations of d and x, i.e., $\tau(d)$ and $\tau(x)$. In the Boolean models, the representation is focused on the absence or presence of terms, leading to binary weights ($\omega \in \{0, 1\}$).

A more effective approach is to use the number of times a term occurs in a document as a weight or measure of importance, $\omega = tf_{t,d}$. The *term frequency (tf)* of term t in document d is denoted by $tf_{t,d}$. As a result, the similarity between the query terms and document terms can be written as Equation 2.3, where t_x is a set of query terms.

$$sim(d, x) := \sum_{i \in t_x} tf_{i,x} \cdot tf_{i,d} \tag{2.3}$$

This equation is biased towards longer documents as they are more prone to have repetitive words. One way to eliminate the unwanted influence of document length is to normalize $\text{tf}_{t,d}$ by the length of a document as shown in Equation 2.4, where $|d|$ is the number of terms in the document d.

$$sim(d, x) := \sum_{i \in t_x} \frac{tf_{i,x} \cdot tf_{i,d}}{|d|} \tag{2.4}$$

Equation 2.4 removes the bias towards long documents, however, all terms in the document are still treated equally. In each document, certain words are more topical and carry more information about the document than others. For example, in the sentence "He went to Westminster Cathedral, the largest Catholic church in the UK.", "Westminster" and "Cathedral" are content-bearing words, whereas "He", "went", and "to" are common linguistic glues that provide no information regarding the topic of the text. [2] In other words, rare terms, which have a lower frequency in the entire corpus

[2] The importance of glue words depends on the domain of the text. In the case of a narrative, the three glues are of vital importance. However, for retrieval purposes, they do not contribute to the relevance of a document.

Bloomberg plans to integrate GPT-style A.I. into its terminal

Bloomberg LP has developed an AI model using the same underlying technology as OpenAI's GPT, and plans to integrate it into features delivered through its terminal software, a company official said in an interview with CNBC.
Bloomberg says that Bloomberg GPT, an internal AI model, can more accurately answer questions like "CEO of Citigroup Inc?", assess whether headlines are bearish or bullish for investors, and even write headlines based on short blurbs.
Large language models trained on terabytes of text data are the hottest corner of the tech industry. Giants such as Microsoft and Google are racing to integrate the technology into their products, and artificial intelligence startups are regularly raising funds at valuations over $1 billion.
Bloomberg's move shows how software developers in many industries beyond Silicon Valley see state-of-the-art AI like GPT as a technical advancement allowing them to automate tasks that used to require a human.
"Both the capabilities of GPT-3 and the way that it achieved its performance through language modeling wasn't something that I expected," said Gideon Mann, head of ML Product and Research at Bloomberg. "So when that came out, 'OK, this is going to change the way that we do NLP here.'"

Top 10 terms ranked by:

frequency		idf		tf * idf	
the	8	thanksto	2.71	bloomberg	0.40
and	6	bloomberg	2.71	gpt	0.27
of	6	pi	2.70	ai	0.17
that	6	27	2.70	like	0.17
bloomberg	5	chung	2.50	headlines	0.16
to	5	imx500	2.50	integrate	0.16
are	4	terabytes	1.90	software	0.16
gpt	4	automate	1.95	technology	0.16
ai	3	based	1.95	way	0.16
an	3	bearish	1.80	language	0.13

Figure 2.3: The difference between *tf-idf*, *idf*, and *frequency* weighing in a sample news article take from `https://www.cnbc.com`. The corpus consisted of 8 short articles.

of documents, are more indicative of the topic. *Inverse document frequency (idf)* is a way to account for such words. *idf* is denoted in Equation 2.5, where $|D|$ is the total number of documents in the corpus and df_t is the number of documents containing the term t.

$$idf_t := log \frac{|D|}{df_t} \tag{2.5}$$

idf represents the likelihood of a randomly selected document containing the term t. This probability tends to be higher for common terms and lower for rare ones. The log function is employed to align the scale of *idf* with term frequency. By inserting *idf* and the document length normalization into the Equation 2.3, we arrive at a very common weighting in IR, called *tf-idf* (Salton and Buckley, 1988), as shown in Equation 2.5.

$$sim(d, x) := \textit{tf-idf}(d, x) = \sum_{i \in t_x} tf_{i,x} \cdot \frac{tf_{i,d}}{|d|} log \frac{|D|}{df_i} \tag{2.6}$$

Relying solely on the term frequency (*tf*) can lead to stopwords and uninformative terms dominating the similarity computation. Conversely, relying solely on the inverse document frequency (*idf*) can skew the similarity towards extremely rare words, such as typos. It is only through the combination of both that a balanced similarity computation can be achieved. Figure 2.3 shows the top-ranked terms by *tf-idf*, *idf*, and raw term frequencies in a snippet from a news article. Raw frequencies rank mostly stop words, obscuring the article's topic. *Idf* ranks rare but uninformative terms higher. In contrast, if we look at the combination of *tf-idf*, the topic of "bloomberg", "gpt",

and "'ai" is clearly distinguishable.

Over the years, several alternatives to *tf* and *idf* have been proposed. Table 2.1 summarizes the most prominent alternatives in SMART notation (Salton and Lesk, 1965). The mnemonic represents a combination of weights as $ddd.qqq$, where the first triplet gives the document weighting and the second the query weighting. The first letter of each triplet represents term frequency, the second letter document frequency, and the last letter is a form of normalization. For example, $lnn.ltn$ equals to $1 + log(tf_{t,x}) \cdot 1 + log(tf_{t,x}) \cdot log\frac{|D|}{df_t}$.

Table 2.1: Table from Manning et al. (2008), describing the variants of *tf-idf*; *CharLength* is the number of characters in a document.

term frequency			document frequency			normalization		
n	$tf_{t,d}$	natural	n	1	none	n	1	none
b	$\begin{cases}1 & \text{if } tf_{t,d} > 0\\ 0 & \text{otherwise}\end{cases}$	Boolean	t	$log\frac{\|D\|}{df_t}$	idf	c	$\frac{1}{\sum_{i=1}^{M}\sqrt{\omega_i^2}}$	cosine
l	$1 + log(tf_{t,d})$	logarithm	p	$max[0, log\frac{\|D\|-df_t}{df_t}]$	prob idf	b	$\frac{1}{CharLen^a}, a < 1$	byte size
a	$0.5 + \frac{0.5tf_{t,d}}{max_t(tf_{t,d})}$	augmented						
L	$\frac{1+log(tf_{t,d})}{1+log(ave_{t\in d}tf_{t,d})}$	log ave						

A full description of all the possible variations of *tf-idf* is beyond the scope of this thesis. Here, we briefly mention *pivoted document length normalization* (Singhal et al., 1996) due to its importance, and for more detailed information of different types, we refer the reader to Manning et al. (2008).

The addition of *idf* comes at a cost of bias towards documents that contain multiple rare words. For example, with query x ="angry Armadillo" and two documents d_1 ="angry ... Armadillo" and d_2 ="Armadillo ... Armadillo", d_2 will be ranked higher due to the large difference in *idf* of the rare term "Armadillo" in comparison to the common term "angry". To solve this issue, pivoted document length normalization was proposed. The idea is that the first encounter of a query term in a document is more important than the rest, and verbose documents that repeat the same content over and over again should not alter the weight. In other words, in long documents repeated term frequencies should be of lower importance in comparison to short ones. Equation 2.7 shows the document length normalization, where the denominator smoothes the weight of term frequency with higher repetition. k parameter is a free parameter, with larger k the effect of squashing is minimal and smaller values increase the effect. $\frac{|d|}{avg|D|}$ accounts for document length, where if the $|d|$ is large, then the frequency weights become smoother in comparison to shorter documents.

$$lenght\ normalization := \frac{tf_{t,d}}{tf_{t,d} + \frac{k|D|}{avg|D|}} \tag{2.7}$$

By adding this normalization to the previous *tf-idf* formula, Equation 2.8 is obtained.

$$sim(d,x) := \textit{tf-idf}(d,x) = \sum_{i \in t_x} tf_{i,x} \cdot \frac{tf_{i,d}}{tf_{i,d} + \frac{k|d|}{avg|D|}} log \frac{|D|}{df_i} \tag{2.8}$$

2.1.3 Probabilistic Models

Another family of classical methods is the probabilistic models that aim to formulate the relevance problem using a probabilistic framework. The fundamental model in this family is known as *Binary Relevance Retrieval (BIR)* (Roelleke et al., 2009), which gave rise to the popular ranking model of BM25. In this section, we go over the BIR model in detail and motivate the derivation of the BM25 ranking function, which forms the basis for our quantity-aware lexical ranker in Chapter 5.
The probability ranking principle (Robertson and Jones, 1976) indicates that the ranking of documents is based on their posterior probability of relevance $p(r|\tau(d))$, which has to be estimated as accurately as possible from the data available to the system. Here, r is a Bernoulli variable, referring to relevance, where $r = 1$ indicates a relevant document and $r = 0$ indicates an irrelevant one. The posterior probability of relevance $p(r|\tau(d))$ is essentially hard to estimate, as it depends not only on the search terms in the query and document but also on the search domain and user interpretation. However, it can be estimated using probability theory with a few assumptions regarding the variables and their dependencies.

(Assumption 1): relevance of any given document is independent of any other documents.

This assumption is common in most IR models, since the ranking of each document is done separately from the other documents in the collection.
Our goal is to estimate a ranking based on $p(r|\tau(d))$. In the following, we describe the steps to simplify this computation. By representing $p(r = 1|\tau(d))$ as x and applying the monotonic transformation $\frac{x}{1-x}$, we can reformulate $p(r = 1|\tau(d))$ as the ratio of the probability of relevance to non-relevance, expressed in Equation 2.9.

$$\frac{p(r = 1|\tau(d))}{p(r = 0|\tau(d))} = \frac{p(r = 1|\tau(d))}{1 - p(r = 1|\tau(d))} \tag{2.9}$$

Since $\frac{x}{1-x}$ is a monotonic transformation, switching from x to $\frac{x}{1-x}$ does not impact the ranking result. By using the Bayes rule after the transformation, $p(r|\tau(d))$ is reformulated in Equation 2.10.

$$sim(d,x) := p(r|\tau(d)) = \frac{p(r = 1|\tau(d))}{p(r = 0|\tau(d))} = \frac{p(\tau(d)|r = 1)p(r = 1)}{p(\tau(d)|r = 0)p(r = 0)} \tag{2.10}$$

The ratio of priors $\frac{p(r=1)}{p(r=0)}$ is constant for all documents and does not affect the final ranking, therefore, it can be disregarded. Consequently, the probability of relevance is estimated by computing

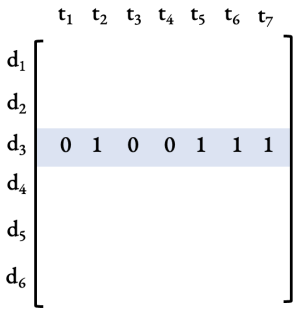


Figure 2.4: Document representation, where a Bernoulli variable defines a presence or absence of a term, also referred to as one-hot encoding (*Assumption 2 and 3*).

$p(r = 1|\tau(d))$ and $p(r = 0|\tau(d))$ and is shown in Equation 2.11 (Robertson and Zaragoza, 2009).

$$p(r|\tau(d)) \sim \frac{p(\tau(d)|r = 1)}{p(\tau(d)|r = 0)} \tag{2.11}$$

To compute these probabilities, we require a definition of τ. To arrive at the document representation, two other assumptions are required:

(Assumption 2): terms are either present or not present, frequency is disregarded.

(Assumption 3): all terms are mutually independent.

Based on these assumptions, a document is modeled as a collection of Bernoulli random variables representing terms in a given vocabulary. These variables indicate if the term t is present or absent in a document $\tau(d) = d_t$. It is worth noting that based on Assumption 3, the random variables d_t are mutually independent, e.g., the presence of the token "Barack" is independent of "Obama", which ignores the association among words. As a result, each document is represented with a binary vector in the size of the vocabulary, where the active dimensions indicate the presence of a term. This form of document representation is also referred to as *one-hot* encoding, since only certain values in a vector are activated while the rest remain zero (similar to the Boolean models from Section 2.1.1)

An example of the vector representation of a document with a Bernoulli variable for the limited vocabulary of five terms is shown in Figure 2.4.
With the independence assumption, the ranking formula (Equation 2.11) is reformulated as products over individual token probabilities, as shown in Equation 2.12. The product goes over all the tokens in the vocabulary, regardless of whether they appear in a document or not. Nevertheless, we can separate the computation for tokens that are present from the absent tokens. To simplify the notation, we define p_t as the probability that term t is present in d given that d is relevant ($p_t = p(d_t = 1|r = 1)$) and g_t if the document is not relevant ($g_t = p(d_t = 1|r = 0)$). Re-

spectively, $1 - p_t$ is the probability of tokens that do not appear in a relevant document and $1 - g_t$ is the probability of tokens that do not appear in an irrelevant document. Consequently, a single product over the entire vocabulary splits into two independent components: one for tokens present in the document and another for those absent.

$$p(r|\tau(d)) \sim \frac{p(\tau(d)|r=1)}{p(\tau(d)|r=0)} \sim \frac{\prod_t p(d_t|r=1)}{\prod_t p(d_t|r=0)} = \prod_{t \in d} \frac{p_t}{g_t} \prod_{t \notin d} \frac{1-p_t}{1-g_t} \tag{2.12}$$

(Assumption 4): the empty document $\vec{0}$, without any tokens, is equally likely to be relevant or non-relevant $p(\vec{0}|r=1) = p(\vec{0}|r=0)$.

Based on *Assumption 4*, ratio of $p(\vec{0}|r=1)$ and $p(\vec{0}|r=0)$ is equal to one, $\prod_t \frac{1-g_t}{1-p_t} = 1$. The nominator is the probability of an empty document occurring in a relevant class and the denominator is the probability of an empty document under the irrelevant class. We can use this property to simplify Equation 2.12 to Equation 2.13, by multiplying this ratio. This multiplication simplifies and isolates the computation to only the tokens in a document.

$$p(r|\tau(d)) \sim \prod_{t \in d} \frac{p_t}{g_t} \prod_{t \notin d} \frac{1-p_t}{1-g_t} \prod_t \frac{1-g_t}{1-p_t} = \prod_{t \in d} \frac{p_t(1-g_t)}{g_t(1-p_t)} \tag{2.13}$$

When provided with relevance judgments for a sample of documents and queries, we can estimate p_t and g_t from the data. Since these probabilities are estimated under the Bernoulli assumption, we can employ maximum likelihood estimation to estimate p_t. This involves determining the percentage of relevant documents that contain token t, as shown in Equation 2.14. $|D(r=1)|$ is the total number of relevant documents (in the entire corpus) and $|D_t(r=1)|$ are the relevant documents (from the entire corpus) containing the term t. 1 is added to avoid division by zero.

$$p_t := \frac{|D_t(r=1)| + 0.5}{|D(r=1)| + 1} \tag{2.14}$$

If we apply the same logic for g_t we arrive at a similar function in Equation 2.15.

$$g_t := \frac{|D_t(r=0)| + 0.5}{|D(r=0)| + 1} \tag{2.15}$$

However, in most cases, relevance judgments are not available to search engines and a way to estimate them without annotated data is beneficial. To this end, additional assumptions are needed:

(Assumption 5): if a token is not part of the query, it is equally likely to occur in relevant and non-relevant documents.

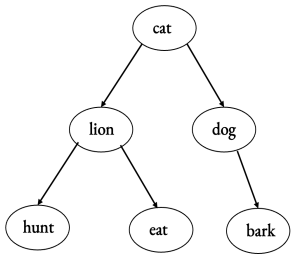


Figure 2.5: A hypothetical example of tree dependence model.

(Assumption 6): on average, a token in the query will occur in half of the relevant document, $p_t = 1 - p_t$.

(Assumption 7): For each query, only a tiny fraction of the documents in the corpus are relevant.

Assumption 5 restricts the product in Equation 2.13 to terms in the query. In accordance with *Assumption 7*, the non-relevant set comprises the majority of documents and can be estimated using the entire collection: $g_t \sim \frac{|D_t|}{|D|} = \frac{|D_t|+0.5}{|D|+1}$. As a result, we arrive at Equation 2.16, where the final fraction is similar to the *idf* weighting in Equation 2.5. This ranking function approximates the summation of the *idf* score of terms present in both the query and document (Croft et al., 2009).

$$p(r|\tau(d)) \sim \prod_{t \in d} \frac{p_t(1-g_t)}{g_t(1-p_t)} = \prod_{t \in d \wedge x} \frac{p_t(1-g_t)}{g_t(1-p_t)} = \prod_{t \in d \wedge x} \frac{(1-g_t)}{g_t} = \prod_{t \in d \wedge x} \frac{|D| - |D_t| + 0.5}{|D_t| + 0.5} \tag{2.16}$$

Equation 2.16 is a classical equation for retrieval, containing many assumptions that are inherently false in natural language. As a result, various improvements have been proposed to alleviate specific assumptions and bring the ranking function closer to real-world scenarios. One of the most problematic ones is *Assumption 3* regarding word independence. An example that contradicts this assumption is the pair "Hong" and "Kong", where the occurrence of one is strongly dependent on the other (Manning et al., 2008). Van Rijsbergen (1979) tackle this assumption by modeling term dependencies as a spanning tree, where the edge weights are the mutual information between terms. Subsequently, the appearance of each term depends on the appearance of its parent in the tree. An illustration of such a tree structure is depicted in Figure 2.5. In this example, the probability of the sentence "cat dog barks" is computed as $p(cat\ dog\ barks) = p(cat)p(dog|cat)p(bark|dog)$.

The tree dependence model is one of the many methods for modeling dependencies among terms. However, despite more realistic modeling of term dependencies, not much gain came from such models and BIR turned out to be the basis for one of the most effective and popular ranking algo-

rithms, known as *BM25* (Robertson and Jones, 1976).

BM25 alters the BIR model in several ways. The most notable one is adding term frequencies to the model, where d_t is no longer a binary value but a count of occurrence of term t in document d. Moreover, BM25 assumes a hidden property in each document as *eliteness*. If a term is deemed elite, it signifies that the document revolves around the concept represented by that term, hence, the term tends to occur frequently in the document. Conversely, non-elite tokens occur at an expected rate by chance, as they do not significantly contribute to the document's topic. BM25 captures these frequency dynamics, including the concept of eliteness, through two Poisson distributions (Harter, 1975), as shown in Equation 2.17. The expected frequency of the term t is denoted by $\mu_{0,t}$ and $\mu_{1,t}$, and el denotes the eliteness.

$$p(d_t) \sim p(el = 1)\frac{e^{-\mu_{1,t}}\mu_{1,t}^{d_t}}{d_t!} + p(el = 0)\frac{e^{-\mu_{0,t}}\mu_{0,t}^{d_t}}{d_t!} \tag{2.17}$$

The parameters in Equation 2.17 are estimated using maximum likelihood. However, the two Poisson model was originally designed for corpus analysis and not retrieval. In retrieval systems, it is crucial to consider the probability of relevance. Robertson and Jones (1976) came up with an approximation to the two Poisson models, conditioned on the relevance. This approximated relevance model estimates $sim(d, x) = \frac{p(r=1|\tau(d))}{p(r=0|\tau(d))}$ and is known as BM25 (Robertson and Jones, 1976), often referred to as Okapi weighting, after the system in which it was initially implemented.
BM25 weighting is shown in Equation 2.18, where dl is the length of a document and $avgdl$ is the average length of documents in the corpus. In addition to t_x denoting all the terms in the query, t_d denotes the terms in a given document d. As a result, the summation computes the heuristic score for terms shared between the document and the query. x_i denotes the frequency of term i in query x. The parameter b controls the impact of length normalization, with $b = 0$ indicating no length normalization and $b = 1$ representing the highest normalization level. A commonly used value for b is 0.75. On the other hand, the constant k_1 governs how the term frequency weight adjusts as the frequency increases. If $k_1 = 0$, the term frequency component would be disregarded, and only the presence or absence of the term would matter. Conversely, for large values of k_1, the term weight would increase almost linearly with frequency. A typical value for k_1 is 1.2, resulting in a non-linear effect where, after three or four occurrences of a term, additional occurrences have minimal impact. Similarly, the constant k_2 plays a similar role in the weight of query terms, with values ranging from 0 to 1000. However, the performance is less sensitive to k_2 since query term frequencies are typically lower and less variable compared to d_i. For optimal performance, it is advisable to tune these parameters to fit the characteristics of the underlying corpus.

$$sim(d, x) \sim \sum_{i \in t_d \wedge t_x} \frac{d_i(1 + k_1)}{d_t + k_1((1 - b) + \frac{b.dl}{avgdl})} log \frac{|D| - |D_i| + 0.5}{|D_i| + 0.5} \cdot \frac{x_i(1 + k_2)}{x_i + k_2} \tag{2.18}$$

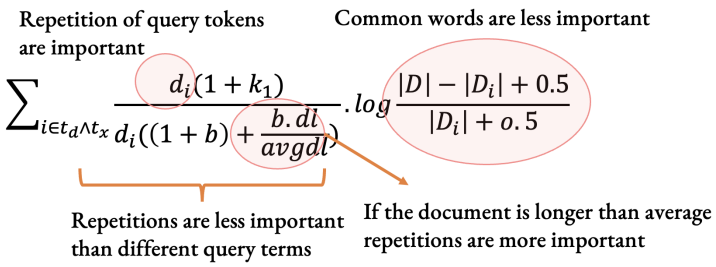


Figure 2.6: BM25 document ranking explained.

We will not go into the derivation of Equation 2.18. Nevertheless, by looking closely at the equation, most parts should be evident from the findings of previous sections. The log term of the equation is the *idf* weight as shown in Equation 2.16. Fractions containing k_1 and k_2 are squashed versions of term frequency in the query and document. Figure 2.6 identifies and depicts different parts of the equation and the intuition behind them.

Even with advances in IR and neural models, BM25 is still considered a powerful baseline. A major advantage of this heuristic function is its simplicity and efficiency. Moreover, it does not require any relevance judgment annotations as all the components of the equation are computed heuristically using corpus statistics, making BM25 work out of the box on any document collection. Because of these advantages, classical methods, particularly BM25, are standard in widely used open-source search software. These open-source search engines are essential because they offer freedom, data privacy, and the ability to be self-hosted, making them popular in both industry and academic settings [3]. Some of the commonly used open-source search engines and libraries are:

- *Apache Lucene:* [4]This high-performance text search engine library in Java offers features for indexing and searching, as well as spell checking and tokenization capabilities. It is widely used in the implementation of many open-source search engines. Many open-source search engines use this library in their implementation.

- *Apache Solr:* [5] This popular open-source enterprise search platform, written in Java, is built on top of the Apache Lucene search library. It offers a wide range of features for searching and indexing documents in various formats. Solr can efficiently handle large volumes of data and is easily integrated with other systems and applications.

- *Elasticsearch:* [6] This is a distributed search and analytics engine built on top of Apache Lucene. It supports customizable text analysis and tokenization pipelines and provides data visualiza-

[3] With advances of the neural models these search engines tend to include some form of vector database as well as the classical techniques, we discuss more about this topic in Section 2.2.5.2.

[4] https://lucene.apache.org/(lastaccessed:02.05.2024)

[5] https://solr.apache.org/(lastaccessed:02.05.2024)

[6] https://www.elastic.co/(lastaccessed:02.05.2024)

tion tools and interactive dashboards. It is widely used by various organizations, including e-commerce sites, news organizations, and government agencies.

- *OpenSearch:* [7] This is an open-source, community-driven search suite derived from Elasticsearch. It builds on the capabilities of Elasticsearch, however, unlike Elasticsearch is governed under the Apache 2.0 license, ensuring that the software remains free and open-source.

It is worth noting that all the classical retrieval models discussed here ignore numerical proximity in their computations. Most classical methods depend on frequency-based index structures, which treat quantities the same as other tokens in the text. When ranking a query that includes quantities, classical models search for exact matches in the document collection, overlooking variations in values, differences in surface forms, value standardization, and numerical proximity. For example, in the query "a mirror with height more than 90 cm", the frequency of the value "90" is not a reliable indicator of relevance. In Chapter 5, we alleviate some of these shortcomings with the help of a dedicated index structure for quantities.

2.2 Neural Retrieval

With the development of machine learning approaches and the availability of large annotated datasets, the retrieval community moved away from the unsupervised realm of heuristic and probabilistic models to supervised learning approaches, e.g., learning to rank (Li et al., 2008; Li, 2011). The idea behind supervised systems is to classify documents into relevant and non-relevant classes, based on a set of lexical and semantic features. The earlier models mainly used a set of hand-crafted features for document and query representation, requiring manual feature engineering. The rise of neural networks paved the way for developing more capable text retrieval systems, which no longer require hand-crafted features.

Going back to Equation 2.1 as our universal similarity function, neural models aim to learn an elaborate representation of a document, $\tau(d)$, and query, $\tau(d)$, by looking at document and query pairs containing relevance judgments, e.g., 1 for relevant and 0 for non-relevant. These representations are mostly low-dimensional vectors (called *dense vectors* or *embeddings*), hence, the name *dense retrieval*. In contrast to the vector space model, where documents were represented by all the terms in the vocabulary, these dense vectors have a much lower dimension and do not correspond to explicit terms, but rather aim at capturing semantic characteristics. The similarity metric, sim, however, in most cases, remains the same as the classical methods, i.e., cosine similarity. Such a retrieval paradigm is referred to as Neural Information Retrieval (Neural IR) (Mitra and Craswell, 2018). In the majority of recent models, the dense representation comes from some form of a language model, more specifically a transformer-based language model. Therefore, to understand the internal architecture

[7] https://opensearch.org/(lastaccessed:02.05.2024)

of the neural model, one first needs to understand the building blocks of such models. In the following, we look at a brief history of how the representation of documents evolved from basic *word embeddings* to neural language models and transformers(Vaswani et al., 2017). Equipped with an understanding of transformers and their capabilities in representation learning, we move on to different types of neural retrieval systems, which adapt these models to ranking problems.
Embeddings and language models discussed here treat quantities the same as other textual tokens. Although recent efforts have been made to create dedicated embeddings and language models for quantitative information, these specialized models typically have limited representation power and are not commonly used in IR models. We discuss some of these dedicated models in Chapter 3. In this section, however, we focus exclusively on the prominent methods used to create query and document representations in IR models.

2.2.1 Word Embeddings

One of the challenges in NLP is to obtain dense representations of words, sentences, and documents. These representations are referred to as embeddings. Word embeddings, as the name suggests, focus on generating dense representations from words (terms). Unlike the one-hot representation used in classical models, which ignores dependencies between words, dense representations aim to encode the semantics of words in context. In a one-hot encoding, words are treated independently, and similarities between words such as "hotel" and "motel" are not considered. One approach to capturing these semantic relationships is to represent a word based on the words it typically co-occurs with. There are several well-known word embeddings based on this idea, such as Word2Vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014), and FastText (Bojanowski et al., 2017). [8]
These embedding methods learn a static embedding for each word in the vocabulary and are also referred to as non-contextualized embedding. A key drawback of these approaches is that the representation for each word remains static, irrespective of the context in which it occurs. For example, in sentences "He attends a play" and "He wanted to play bridge", the embedding of the word "play" is the same, despite the semantics being significantly different. Moreover, in retrieval settings, a representation of a sequence of words or a document is required, which word embeddings do not provide on their own. Adding and averaging them across a document is one solution, however, such combinations usually result in poor performance, mix-up of static representations, and loss of information about the relations of the words. It is more suitable to have a direct and unified representation of documents and queries in the same space, which takes the entire context into account. In context-aware models, embeddings are a function of both a word and the context, where the context is often encoded using some form of a language model.

[8] We do not go deep into the topic of word embedding and different architectures, but point the reader to a survey by Sezerer and Tekir (2021).

2.2.2 Early Language Models

Language modeling in NLP refers to the task of predicting the word or token that comes next, e.g., in the sentence "The teacher cleaned the ___ ", a language model fills the blank with potential words such as "blackboard" or "desk". Formally, given a sequence of tokens $t^{(1)}, t^{(2)}, ..., t^{(m)}$, a language model computes the probability distribution of the next token $p(t^{(m+1)}|t^{(m)}...t^{(1)})$. Another way to phrase the task is to view a language model as a system that assigns a probability to a piece of text. These probabilities are decomposed using the chain rule into a product of the probabilities of predicting the next word, expressed as $p(t^{(1)}...t^{(m)}) = \prod_t^{Vocab} p(t^{(m)}|t^{(m-1)}...t^{(1)})$ (Eisenstein, 2019). Here, *Vocab* refers to all terms in a given vocabulary.

The traditional way to estimate next-word probabilities is using *n-gram language models*. *n-grams* are chunks of consecutive words, e.g., "teacher" is a unigram, "the teacher" is a bigram and "the teacher cleaned" is a trigram. In n-gram language models, the frequencies of n-grams in a given corpus are analyzed to construct a probability model (Jurafsky and Martin, 2009). With advances in neural networks, language models have moved from the discrete space of n-grams to a continuous space with potentially fewer parameters.

The initial neural network-based language model was introduced by Bengio et al. (2000, 2003). This model employed a feed-forward network architecture, where a fixed window of tokens from the text serves as the input to predict the next token. [9]

While feed-forward networks address the sparsity issue and demand less storage, the window size is still fixed and word order is disregarded. Additionally, fully connected networks have a considerable number of parameters and are costly to train.

Recurrent neural networks (RNNs) (Hochreiter and Schmidhuber, 1997; Schmidt, 2019) revolutionized language modeling by enabling the processing of inputs of arbitrary length, facilitating larger context utilization and weight sharing. In RNNs, the hidden layer is fed into the network recurrently, hence, weights are shared and the model size is smaller. Consequently, in RNNs, there are only two parameter matrices (disregarding the embedding): one for multiplying the input embeddings W_e and one for updating the hidden state W_h.

The computation of the probability of the next word with a simple RNN is summarized in Equations 2.19. The prediction of a token at time-step n is not solely influenced by the embedding of the current token $e^{(n)}$. Instead, it is combined with the hidden state of the previous cell $h^{(n-1)}$ to produce the current hidden state $h^{(n)}$.

$$\begin{aligned} e^{(n)} &= Et^{(n)} + b_1 \\ h^{(n)} &= f(W_h h^{(n-1)} + W_e e^{(n)} + b_2) \\ y^{(n)} &= softmax(W_1 h^{(n)} + b_3) \in \mathbb{R}^{|Vocab|} \end{aligned} \tag{2.19}$$

[9] Here, we switch to token, since the language models based on neural network employ various tokenization techniques, from words, to sub-words or characters.

In practice, RNN models have difficulty remembering long contexts, due to the *vanishing gradient* and *exploding gradient* problem (Pascanu et al., 2013). Although the exploding gradient problem can be addressed by clipping gradients at some maximum value, a vanishing gradient requires a change in architecture. Later, memory units such as *Long Shot Term Memory (LSTMs)* (Gers et al., 2000; Hochreiter and Schmidhuber, 1997) and *Gated Recurrent Unit (GRUs)* (Cho et al., 2014b) are proposed to better preserve information over many time steps. A memory unit is a neuron with a recurrent self-connection that has an extra hidden vector as *cell state*. The cell aims to store long-term information. The parameters of the memory unit decide how often the cell state should be updated or maintained in each time step (Eisenstein, 2019).

An extension to RNNs is *bi-directionality*, where both preceding and succeeding contexts are utilized for prediction. This is achieved by employing two separate RNNs: one reads the sentence from left to right, and the other, with different parameters, reads from right to left. The combined representation of each token is then obtained by concatenating the hidden states of both RNNs. The computation of the hidden state at time n is shown in Equation 2.20, where RNN_{FW} and RNN_{BW} represent the forward and backward networks, respectively, and $h^{(n)}$ denotes the concatenation of the hidden layers in both directions.

$$\begin{aligned} \vec{h}^{(n)} &= \text{RNN}_{FW} = (h^{(n-1)}, t^n) \\ \overleftarrow{h}^{(n)} &= \text{RNN}_{FW} = (h^{(n+1)}, t^n) \\ h^{(n)} &= [\vec{h}^{(n)}; \overleftarrow{h}^{(n)}] \end{aligned} \tag{2.20}$$

A combination of bi-directionality and LSTMs gave rise to one of the earliest works that created contextualized embeddings using language modeling. *ELMo (Embedding from Language Models)* (Peters et al., 2018) learns contextualized word representation with multiple layers of bi-directional LSTMs. An important and innovative part of ELMo's architecture is that, unlike RNN models, all internal layer representations of ELMo's architecture along with the last hidden state contribute to the final embedding. A linear combination of all hidden layers creates the final embedding, which is used for various downstream tasks.

The input of ELMo is a sentence and as a result, it generates not only token representation but also sentence embeddings. To use these embedding for downstream tasks, such as sentiment analysis, question answering, or named entity recognition, a weighted combination of the layer-wise representation is computed.ELMo showed that by pre-training on a large corpus of text for language modeling, the internal representations of a language model are capable of encoding semantic and grammatical dependencies, which can be utilized in a variety of downstream applications. Moreover, different layers of deep bidirectional RNNs encode diverse types of information. Hence, leveraging this variability by combining layers is essential. For instance, lower levels of a deep LSTM are adept at handling lower-level tasks like dependency parsing (Hashimoto et al., 2017), while the top layer of an

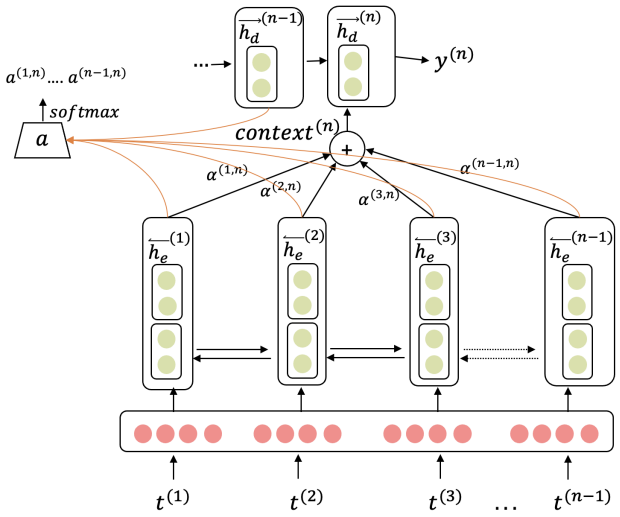


Figure 2.7: Attention mechanism for RNN encoder-decoder.

LSTM specializes in learning representations of word sense (Melamud et al., 2016). This paradigm becomes more prominent as we move on to transformer-based language models.

2.2.3 Transformers

Recent language models and neural IR architectures have shifted from using RNNs to leveraging transformers for document representation (Vaswani et al., 2017). In this section, we provide a detailed introduction to transformers, as understanding the inner workings of this model and how it captures term interactions is crucial for gaining insight into the representation of documents and queries in modern IR models.

In RNNs, the linear order of words is forced by the model, which is not always desirable. They are computationally slow and resource-intensive to train on large text corpora. Moreover, local relationships are captured well, but as the sequence length grows, it becomes harder to learn long-term dependencies. In 2015, Bahdanau et al. (2015) introduced *attention* to solve exactly this problem in machine translation. Although the original paper rarely uses the term attention, subsequent works adopted this terminology. The core idea behind attention is to consider the hidden states of the entire input sequence when making predictions, rather than relying solely on the final hidden state that represents the entire input. Then the task of the attention mechanism is to learn how much to *attend* and give focus to each hidden state. The concept is best described using Figure 2.7, where attention is applied on top of a bi-directional RNN to compute a weighted linear combination of all hidden states. The figure is created with the use-case of machine translation in mind, in which an encoder (bi-directional RNN) creates a representation of an input sentence, and a decoder uses the representation to generate tokens of the translation. The *encoder-decoder* architecture, where a network encodes the input into a compact representation and a decoder generates text based on this condensed representation, is widely used in literature.

Given the hidden states of the decoder as h_d, at each time-step n, the attention mechanism decides the amount of attention to be paid to each hidden encoder unit h_e, by calculating the attention weight a as a function of all encoder hidden states and the previous representation $s^{(n-1)}$. More formally, the attention weights for hidden state j at time-step i are defined in Equation 2.21, where *FFNN* is a feed-forward network. In other words, $a^{(i,j)}$ determines how much the decoder should pay attention to $h_e^{(j)}$ while computing the output token at time-step i.

$$\begin{aligned} a^{(i,j)} &:= \frac{\exp(A^{(i,j)})}{\sum_{k=1}^{n} \exp(A^{(i,k)})} \\ A^{(i,j)} &:= \mathit{FFNN}(h_d^{(i-1)}, h_e^{(j)}) \end{aligned} \tag{2.21}$$

The entire context is captured by the *context vector*, which is the linear combination of hidden states based on the attention weights, a, computed as shown in Equation 2.22.

$$context^i := \sum_{j=1}^{n} a^{j,i} h^{(j)} \tag{2.22}$$

The described attention mechanism transfers information from the encoder to the decoder. In contrast, *self-attention* computes attention within a single input and was first introduced in 2017 by Vaswani et al. (2017) along with the transformer architecture for machine translation tasks. These concepts form the foundation of current language models.

Self-attention works with three sets of vectors: queries $q_u^{(n)} \in \mathbb{R}^k$, keys $k^{(n)} \in \mathbb{R}^k$, and values $\nu^{(n)} \in \mathbb{R}^\nu$. These vectors are computed for each token by multiplying the token embedding by three matrices that are learned during the training process, namely W_{Q_u}, W_K, and $W_\mathcal{V}$. The basic idea is that the query vector of each token is compared against the key vectors of all other tokens for compatibility. The greater the compatibility, the more influence the corresponding value will have on the output representation of the query token. For each token, a query, key, and value are computed as a linear transformation of their token embeddings. In practice, the queries, keys, and values of all tokens are packed together into matrices Q_u, K, and $\mathcal{V}$.
If an embedding for token n is denoted as $e^{(n)}$ then $q_u^{(n)} = W_{Q_u}.e^{(n)}$, $\nu^{(n)} = W_\mathcal{V}.e^{(n)}$ and $k^{(n)} = W_K.e^{(n)}$. The dot product between the query vector of a token, i.e., $q_u^{(i)}$, and the key vector of another token, i.e., $k^{(j)}$, determines their compatibility. e.g., $q_u^{(i)} k^{(j)}$ determines how compatible or important token j is to token i. This is referred to as *dot-product attention* that computes the attention weights a as a set of unnormalized weights. The effect of dot-product attention suffers for large dimensionality of key, query, and value embeddings (d_k), and authors attribute this to softmax gradients vanishing for high dimensional inputs. Dividing by standard deviation, $\sqrt{d_k}$, helps counteract this problem. As a result, the weights are re-scaled by dividing by the dimensionality of query and key vectors $\sqrt{d_k}$. For normalization into non-negative values that sum to one, a softmax layer is

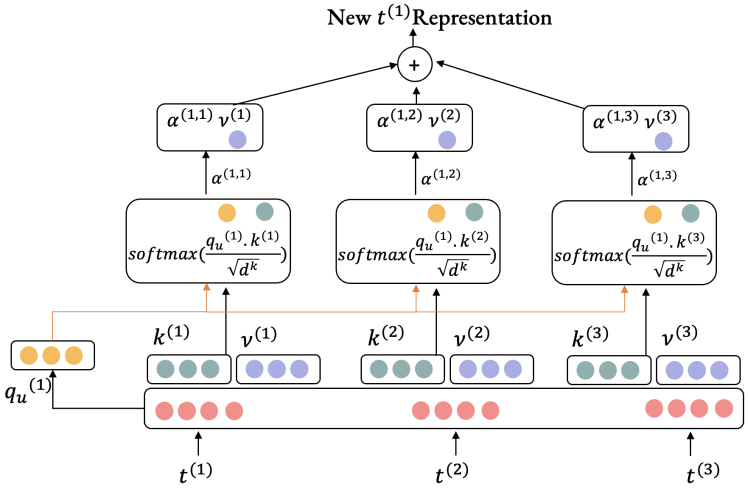


Figure 2.8: Self-attention computation for a single token based on the representation of others, in this example the sentence has only three tokens.

applied, e.g., $a^{(i,j)} = softmax(\frac{q_u^{(i)}.k^{(j)}}{\sqrt{d_k}})$. The computation of the attention weights in a vector, a, is shown in Equation 2.23.

$$a := softmax(\frac{Q_u K^T}{\sqrt{d_k}}) \tag{2.23}$$

The output of the attention network is an updated representation for each token, based on the influence of the entire input, e.g., $\sum_{j=1}^{n} a^{(i,j)} v^j$ is the new representation for token i based on all the other tokens in the input sequence. The attention computation is shown in Equation 2.24.

$$Attention(Q_u, K, \mathcal{V}) := softmax(\frac{Q_u K^T}{\sqrt{d_k}}) \cdot \mathcal{V} \tag{2.24}$$

Figure 2.8 goes over the computation for a single token in a sequence consisting of only 3 tokens. Based on the coloring scheme it is visible how the final representation of a token $t^{(1)}$ becomes the combination of different value vectors. The amount of participation in the final representation is selected by the interaction between the query vector $q_u^{(1)}$ and the key vectors of all other tokens. A single attention head computes one linear combination, while multiple attention heads learn several representations simultaneously. For a set of m attention heads, a set of m learned projection matrices are defined $W_{Q_{u_i}}$, W_{K_i}, and $W_{\mathcal{V}_i}$, where i is indexing over the heads. The *multi-head* attention block computes the attention function m concatenates the outputs, and multiplies them by another weight matrix (W_O) to project the multiple representations back to the original dimensionality. The computation of multi-head attention is defined in Equation 2.25.

$$\begin{aligned} &Multi\text{-}Head(Q_u, K, \mathcal{V}) = Concat(head_1, ..., head_m)W_O \\ &head_i := Attention(Q_{u_i}, K_u, \mathcal{V}_u) \end{aligned} \tag{2.25}$$

Positional encoding with d=4

sequence	pos	i=0	i=0	i=1	i=1
The	0	PE_{00}=sin(0)=0	PE_{01}=cos(0)=1	PE_{02}=sin(0)=0	PE_{03}=cos(0)=0
teacher	1	PE_{10}=sin(1/1)= 0.84	PE_{11}=cos(1/1)= 0.54	PE_{12}=sin(1/100) =0.009	PE_{13}=cos(1/100) =1
cleaned	2	PE_{20}=sin(2/1)= 0.91	PE_{21}=cos(2/1)= -0.42	PE_{22}=sin(2/100) =0.019	PE_{23}=cos(2/100) =0.99
the	3	PE_{30}=sin(3/1)= 0.14	PE_{31}=cos(1/1)= -0.99	PE_{32}=sin(3/100) =0.029	PE_{33}=cos(3/100) =0.99

Figure 2.9: Positional encoding matrix for "the teacher cleaned the", with dimension $D = 4$.

Unlike recurrent models, the input in a transformer is processed in parallel, significantly reducing computation time. However, processing inputs in parallel means the order of words is lost. To retrain the word order, *positional encodings* are added to the token embeddings to inject some information about the relative or absolute position of the tokens. There are many choices for positional embeddings. The first ones are sine and cosine functions of different frequencies, where the frequency decreases as the dimension in index i (number of tokens in a sequence) increases. Positional encoding functions are shown in Equation 2.26, where pos is the position of the word in a sequence and D is the dimension of the embedding.

$$\begin{aligned} PE_{(pos,2i)} &:= sin(pos/10000^{(2i/D)}) \\ PE_{(pos,2i+1)} &:= cos(pos/10000^{(2i/D)}) \end{aligned} \tag{2.26}$$

In Equation 2.26, the positions of the even numbers follow a sinus and the positions of the odd numbers follow a cosine curve. An example of the positional encoding for a sequence with four tokens is shown in Figure 2.9, where the encoding value is the function of its position in the input sequence. In fact, the positional encoding matrix would be the same for any four-letter phrase. However, more recent models tend to learn the positional embeddings as part of the model parameter and the sinus and cosine functions are used less commonly. With the help of positional encodings and multi-head attention, a transformer block (Tunstall et al., 2022) is built. The transformer block consists of two sublayers: the first is multi-head attention, and the second is a feed-forward network. After each sublayer a *residual connection* and *layer normalization* (Ba et al., 2016) are performed ($LayerNorm(x + Sublayer(x))$). Here, $Sublayer(x)$ denotes either the multi-head attention mechanism or the feed-forward network.

Figure 2.10 shows the entire transformer architecture. In the first sublayer, *Add & Norm* denotes the residual connection plus layer normalization. Layer normalization adjusts each token vector to have a mean of zero and a standard deviation of one. The second sublayer is a position-wise feed-forward network, with fully connected layers applied independently to each input position, followed by another Add & Norm block. Each grey block, comprising of multi-head attention and feed-forward

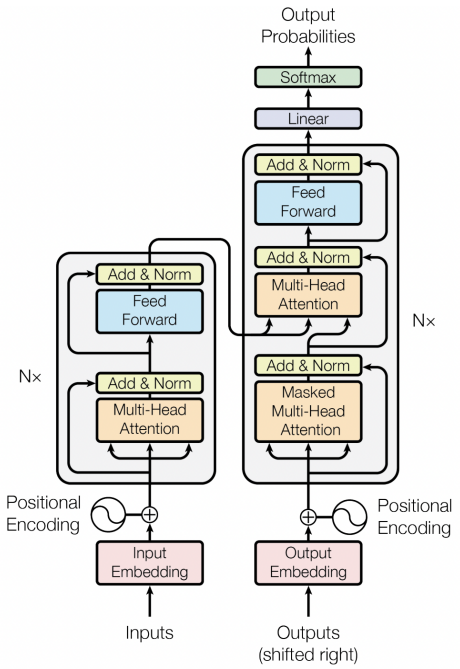


Figure 2.10: The transformer architecture. Image taken from Vaswani et al. (2017).

network, is stacked N times to form the transformer encoder.

The decoder block resembles the encoder but with a few distinctions. The first multi-head attention block is replaced with a *masked multi-head attention*. A mask preserves the temporal dependencies in the output sequence, ensuring predictions at each time step rely only on prior known outputs. Such a model is called *auto-regressive* (Graves, 2013), generating the next token based on the previous context. A mask is simply an upper triangular matrix, with $-\infty$ above the diagonal and zero under it, which is added to the attention weights before the softmax. The softmax pushes the $-\infty$ values to zero, causing zero attention for tokens after a specified position.
The other multi-head attention block in the decoder computes *cross-attention* between the encoder and decoder. In the encoder, the goal is to create a representation of the input, therefore attention is computed on the input sentence only. However, in the decoder, the aim shifts to generating output conditioned on both the encoder representation and the generated sequence. Cross-attention merges these elements by using queries, keys, and values from different sequences. Queries or selectors come from the previous layer of the decoder. Keys and values come from the final layer of the encoder. As a result, the decoder attends simultaneously to the sequence generated so far and the input. The rest of the decoder module is similar to the encoder, where the last layer is the vector representation for each token in the output sequence. A softmax linear layer converts these vectors to probabilities over the target vocabulary to predict the next token.
Although the initial transformer contains both encoder and decoder blocks, depending on the application only one would suffice. For example, for generating sequences, only a decoder block is enough, or for a representation learning task, an encoder is more suitable.

2.2.4 Transformer-based language models

One of the main research questions in the era of deep learning is how to train effective and task-specific models with limited annotated data. A major milestone for solving this issue comes from transfer learning. This paradigm, first introduced in computer vision, is inspired by the fact that humans can solve novel problems with only a few samples because of the knowledge they have acquired throughout their lives. In transfer learning, the model undergoes a pre-training phase to grasp basic and common features, followed by a fine-tuning stage where this acquired knowledge is applied to target tasks (Han et al., 2021). The pre-training stage is usually an unsupervised objective or a task with an abundance of data, whereas task-specific data are rather scarce.

It was not until language models that transfer learning made its way into NLP. Dai and Le (2015) were the first to take advantage of this paradigm in NLP, followed by the ELMo model. ELMo performed language modeling as pre-training on a large amount of text and only used smaller sets for fine-tuning on specific tasks. Transformers made it possible to speed up the costly pre-training task on ElMo's recurrent architecture and achieve better performance.

Another trend that started with the rise of transformer language models was the shift to *sub-word tokenization* instead of word level or character level tokenization which is more relevant for the traditional models. Sub-word tokenization addresses the limitations of both word-based and character-based tokenization approaches. It mitigates issues like large vocabulary sizes and out-of-vocabulary tokens encountered in word-based methods, while also resolving concerns such as lengthy inputs and the lack of meaningful individual tokens found in character-based approaches. In sub-word tokenization, common words remain unchanged, while infrequent words are broken down into meaningful sub-units. For example, "girls" is split into "girl" and the plural indicator "s". This helps the model detect the root word and shift in meaning by adding suffixes. Some of the popular sub-word tokenization algorithms are *Byte-Pair Encoding (BPE)* that is first introduced in an article in 1994 (Gage, 1994) and later used in machine translation (Sennrich et al., 2016b), WordPiece (Song et al., 2021), and SentencePiece (Kudo and Richardson, 2018). We do not go over the details of sub-word tokenization and refer the reader instead to the respective papers. For all the transformer models, when we refer to a token, a sub-word token is meant.

Transformer-based language models can be classified into two main types: encoder-based and decoder-based models. In this section, we briefly describe each type and highlight their differences. The encoder-based models are more prominent in IR models for learning query and document representations and are discussed in depth. All the neural architectures described later in this chapter use some form of encoder-based language model as their base model.

On the other hand, decoder-based models have gained significant interest in recent years, particularly with the GPT models (Radford et al., 2018). Since one variant of these models is employed as a baseline system in Chapter 4, we also provide a brief description of the decoder-based approach with an emphasis on GPT.

2.2.4.1 Decoder-based Language Models

The decoder of a transformer (decoder part of an encoder-decoder architecture) is designed to generate a sequence conditioning on the previous context, making it suitable for language modeling tasks. To fine-tune the decoder on a specific task, the final softmax linear layer that predicts the next token is replaced by a task-specific head. For example, for sentiment analysis, a linear layer predicting the sentiment is added.
The first model in the line of decoder base language models is *Generative Pretrained Transformer* or *GPT* for short (Radford et al., 2018). The first version of GPT is a transformer decoder with 12 layers, Byte-Pair Encoding as a tokenizer, and trained 7000 unique books, where long spans of continuous text are used as input. GPT is a relatively small model with 117 million parameters. The pre-trained model is then fine-tuned on a variety of tasks, including classification, natural language inference, semantic similarity, and question answering. As expected, pre-training greatly benefits downstream applications. The main contribution of the first GPT is semi-supervised learning for NLP tasks, where unsupervised language modeling is the initial and resource-hungry part of the training. For fine-tuning, the authors added an auxiliary learning objective to get better generalization and faster convergence, where they combine the language modeling objective with a downstream task ($L_{total} = L_{fine\text{-}tune} + \lambda L_{LM}$). The total loss is a combination of fine-tuning loss ($L_{fine\text{-}tune}$) and language modeling loss (L_{LM}), where λ weighs down the language modeling objective.

In 2019, GPT-2 (Radford et al., 2019) was released as a larger model with more parameters and larger hidden units, scaling up to 1.5 billion parameters. In comparison to the previous version, GPT-2 is better at generating convincing and consistent text in natural language. Starting from GPT-2, the GPT model series became controversial as the creators withheld its full version, claiming it to be dangerous in generating fake news and harmful content. Here, the authors introduced task conditioning, where the model is expected to produce different outputs for the same input for different tasks. To this end, the language modeling objective of predicting an output sequence given an input $P(output|input)$ is transformed to condition on the task as well, $P(output|input, task)$. The $task$ is a natural language instruction provided in addition to the input sequence, where the output is conditioned on both. This provided the basis for *zero shot task transfer*, where the model understands and solves a task only based on a given instruction without any training examples.

The GPT-3 model published in the year 2020 with 175 billion parameters marks a breakthrough that is capable of generating text, which is hardly distinguishable from human-written content. Other than the increase in parameter size, an enormous corpus of text data is added to the model training set. Additionally, on-the-fly task learning emerges on tasks that the GPT-3 model is never explicitly trained on, like writing code and SQL queries with only providing one or few examples. This is referred to as *few-shot* and *one-shot* training. Such capabilities come from *in-context learning*, in

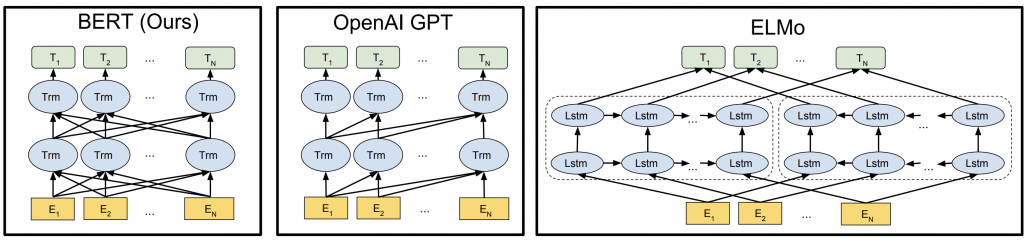


Figure 2.11: Differences in model architectures of BERT, GPT, and ELMo. BERT uses an encoder part of a Transformer, GPT uses a left-to-right decoder transformer and, ELMo uses the concatenation of left-to-right and right-to-left LSTMs to generate features. Image taken from Devlin et al. (2019a).

which the language model develops pattern recognition and other skills helping the model to extend its capabilities to unseen tasks. During in-context learning, the language model receives a prompt that consists of a list of input-output pairs that demonstrate a task. At the end of the prompt, a test input is appended for the model to predict the answer.

More recent variants of GPT models use more sophisticated training paradigms to bring the outputs of the model closer to human preference. GPT-3.5 uses a technique called *Reinforcement Learning from Human Feedback (RLHF)* (Christiano et al., 2017) for better alignment to human preference. In RLHF, the model learns to make decisions or take actions based on feedback provided by humans. Moreover, GPT-3.5 introduces *instruction tuning*, by fine-tuning a language model on a dataset of instructions and corresponding responses. This process helps the model better understand and follow human-provided instructions. GPT-3.5 is also referred to as InstructGPT and is the underlying model for ChatGPT. [10]

GPT-4 (OpenAI et al., 2023) is another model in this series and its main contribution is to have extended the input from text to images and videos as well, i.e., multi-modality. [11]

2.2.4.2 Encoder-based Language Models

A less intuitive way to train a language model is to take the encoder of a transformer without the autoregressive masking. The first and most prominent model in this domain is *Bidirectional Encoder Representations from Transformers (BERT)* (Devlin et al., 2019a). Figure 2.11 shows a comparison between the main three models discussed so far, namely, BERT, GPT, and ELMo. Only BERT representations are conditioned on both the left and right context in all layers. Moreover, BERT and GPT introduce fewer task-specific parameters in comparison to ELMo. While ELMo learns general textual features during language modeling, it freezes these layers during fine-tuning, preventing the flow of information to the embedding layers for downstream tasks. In contrast, BERT and GPT are

[10] ChatGPT from Open AI: `https://chat.openai.com/chat` (last accessed: 02.05.2024)
[11] Given the rapid development in this field, there new language models published almost weekly and this list is not comprehensive.

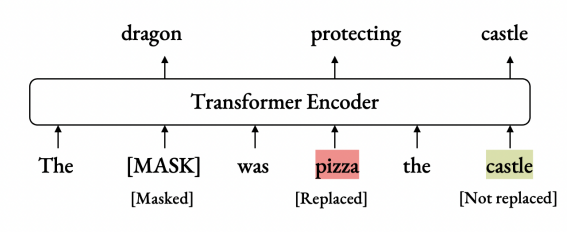


Figure 2.12: Example of MLM for BERT (Devlin et al., 2019a), for an example sentence of "The dragon was protecting the castle".

fine-tuned for downstream tasks by updating all pre-trained parameters, improving performance on specific tasks. An encoder creates representations with bi-directional context, considering both left and right context, unlike traditional language models that only consider th previous context. BERT combines ELMo's bi-directionality, the efficiency of transformers, and GPT's end-to-end training into one model.

To formulate the bi-directionality as a language modeling task, BERT (Devlin et al., 2019a) uses *Masked Language Modeling (MLM)*. The idea is to replace a fraction of the tokens in a sentence with a mask token (`[MASK]`). The model task is to predict the mask tokens based on the entire context. For example, in "The dragon was protecting the castle", random masking results in "The `[MASK]` was protecting `[MASK]` castle". However, if the model is always conditioned to make predictions upon encountering a `[MASK]` token, it does not develop strong representations for unmasked tokens. To introduce more diversity, 15% of the sub-word tokens are randomly selected, which:

- 80% of the time are replaced with `[MASK]`,
- 10% of the time replaced with a random token, and
- 10% of the time are left unchanged.

Consequently, for each sentence, the model is penalized for three different mask variation and optimized using cross-entropy loss.

In addition to MLM, BERT pre-training includes a *Next Sentence Prediction (NSP)* task, training the model to predict if one text chunk follows another. NSP aims to teach relationships between text pieces, useful for tasks like question answering where context differs between question and answer. However, later works argued that this objective is not necessary for learning good quality representations (Liu et al., 2019; Yang et al., 2019).

In addition to different training objectives, BERT introduces additional layers to the token embedding of the default transformer:

1. *Token embeddings:* Each token is encoded as a low-dimensional vector, and two sentences are fed into the model. Two special tokens are added: `[CLS]` at the beginning of the first sentence

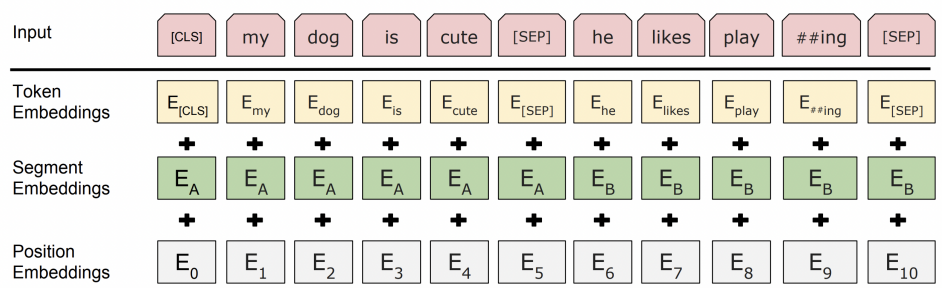


Figure 2.13: Embedding layer so of BERT; Image taken from BERT (Devlin et al., 2019a).

and `[SEP]` at the end of each sentence. `[CLS]` stands for classification, with its final hidden state used as the aggregate sequence representation for classification tasks.
Another way to obtain sentence representation is by max/mean pooling of all token representations from the last layer before softmax. However, the authors argue that `[CLS]` offers better sentence-level understanding due to its use in NSP during pre-training. Averaging all token embeddings gives equal weight to all tokens, including stop words. In contrast, the `[CLS]` vector, computed with self-attention, collects task-relevant information. During pre-training, `[CLS]` goes through a classification layer to detect if two sentences divided by `[SEP]` are consecutive.

2. *Positional embeddings:* Similar to the basic transformer architecture, a position embedding is added to token embeddings to keep track of the positions of tokens in self-attention modules.

3. *Segment embeddings:* A marker indicating Sentence One or Sentence Two is added to each token. This enables the encoder to distinguish between the two text chunks.

These embeddings are shown in Figure 2.13.
BERT was released with two variants based on the model size, as `bert-base` with 12 layers and 768 hidden dimensions and `bert-large` with 24 layers and 1024 hidden dimensions. Both models are trained on the English Wikipedia and BookCorpus. Compared to recent GPT models, BERT is relatively small. However, it still requires four days for pre-training with 64 TPUs, making it impractical for single-GPU or academic pre-training. Despite the resource-intensive training, fine-tuning BERT is fast and easily achievable on a single GPU, making it appealing for many downstream applications, the most notable of which is IR. In contrast to generative models like GPT, which excel at text generation, encoders are adept at capturing representations and features across various output formats. This reason makes them more common in retrieval settings, where the task revolves around searching through vast corpora of text rather than generating new text.

BERT is the first and most well-known encoder-based language model. There have been many variants of it proposed over the years mainly to improve the training procedure. Some of the most prominent variations that are commonly used in IR models and their differences from the base model are

organized in Table 2.2. This list is not exhaustive. Although models such as ALBERT (Lan et al., 2020), ELECTRA (Clark et al., 2020), SpanBERT (Joshi et al., 2020), and XLNet (Yang et al., 2019) have made significant improvements in the development of language models, their findings are less relevant to the IR and are therefore not discussed here. We provide only a brief overview of the main ideas, for detailed information on each model, refer to the respective papers.

Table 2.2: List of the important model variants of BERT.

Model	Summary
1. RoBERTa (Liu et al., 2019)	(1) dynamic masking (2) remove NSP (3) More data (4) larger batches
2. DistilBERT (Sanh et al., 2019)	reduce the size of the model with knowledge distillation
3. SBERT (Reimers and Gurevych, 2019)	dedicated sentence embedding by using Siamese BERT networks

1. *RoBERTa*: To make the BERT model more robust, the authors introduce *dynamic masking*. In contrast to BERT static masking, different parts of the sentences are masked, for different epochs. NSP is also removed from the objectives as it is deemed unuseful. Additionally, more data is incorporated into the pre-training corpus, and the model undergoes longer training sessions with larger batch sizes. For instance, the batch size is increased to 8,000 over 300,000 steps, contrasting with BERT's batch size of 256 and 1 million steps. RoBERTa is a standard substitute for better in many IR models that aim to get a slight performance boost over the base model without a change in architecture.

2. *DistilBERT*: This model does not tweak the BERT architecture or training procedure but tries to reduce its size, by creating a smaller model that replicates the output of BERT in a process called *knowledge distillation*. In knowledge distillation, the knowledge of a large model, a teacher, is distilled to a smaller one, a student, that produces similar output. Knowledge distillation is widely used in IR models to enhance the performance of efficient and small models with the knowledge of the expensive and large ones. We discuss more about this topic in the next section. In the case of BERT the student model learns the probability distribution of the predictions made by the original BERT model. The authors show that with a much smaller network, they can achieve on-par results with the original model.

3. *SBERT*: In order to measure the similarity between two sentences with BERT, one would concatenate them with a `[SEP]` token in between and use the `[CLS]` token to compute their similarity. This process is rather inefficient as it computes attention across all tokens of the first sentence to the second sentence. The cross-attention construction makes BERT unsuitable for semantic similarity search in an efficient manner. *Sentence-BERT (SBERT)* presents a modification of the pre-trained BERT network that uses the siamese architecture of two BERTs that share parameters. SBERT adds a pooling operation to the output of BERT to derive sentence embeddings. Both embeddings and their element-wise differences are concatenated and fed through a classification layer to predict their similarity. Constructs similar to

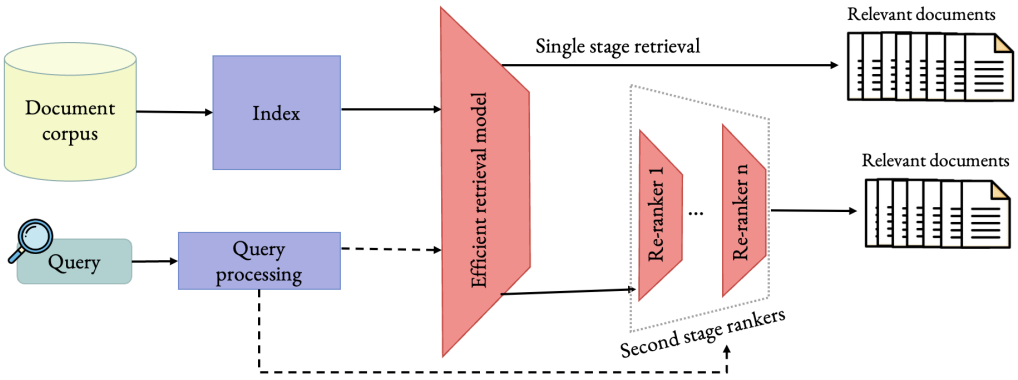


Figure 2.14: Multi-stage architecture of modern IR systems.

SBERT are popular in retrieval settings as bi-encoder architectures, where one BERT model computes the representation for the query and another for the document. We describe such models in a later section.

2.2.5 Retrieval with Language Models

Pre-trained language models changed the landscape of retrieval systems. Their powerful representations have replaced traditional frequency-based methods to represent queries and documents, while still following a similar paradigm. As mentioned at the beginning of this chapter, the goal of a retrieval system is to estimate the relevance of a query x to a document d based on the similarity of their representations $\tau(x)$ and $\tau(d)$ or, in other words, to estimate the probability $p(d|x)$ for documents in the corpus.

In neural retrieval $\tau(x)$ and $\tau(d)$ come from a pre-trained language model. As in traditional methods, the similarity function commonly used to compute the similarity of query and document is cosine or dot product. However, unlike the traditional and probabilistic methods with multiple independence assumptions among tokens and documents, a pre-trained language model captures the interaction among all the tokens with the help of the attention module and their representation is context-dependent. The attention mechanism resolves several limitations of traditional models. However, these dense representations are derived from language models with millions, and sometimes billions, of parameters, rendering them computationally intensive and less interpretable. Additionally, computing dot products among these dense vectors for large corpora with millions of documents is impractical for real-time search engines. To balance efficiency and effectiveness, many modern systems adapt a multi-stage ranking pipeline, depicted in Figure 2.14. The first stage returns an initial set of candidates from the entire corpus with fast and efficient ranking models assisted by an embedding or term-based index. Later ranking stages are more complex and effective with the aim of pruning and improving the final result set. First-stage models are typically designed to have a high recall and return potentially all relevant documents from the entire collections in a short time span.

Whereas, in contrast, the secondary stage ranker emphasizes precision, with the aim to position the most relevant documents at the top (Aumiller et al., 2020; Guo et al., 2022).
The first-stage rankers have long been dominated by lexical models. Due to their simplicity, powerful index structures, and integrability in most open-source search engines, they are good candidates for efficient retrieval. Although they have demonstrated reasonable performance in practice, they still suffer from vocabulary mismatch and do not capture document semantics well by disregarding word order. Moreover, the most important property of the first stage ranker is high recall and lexical models tend to ignore semantically similar results with different vocabulary.
These limitations act as a blocker, preventing second-stage rankers from accessing some of the relevant documents (Chen et al., 2017). On that account, the development in recent years has focused on creating first-stage neural retrieval systems that produce a high-quality ranking in an efficient manner. In contrast to classical methods, the representations from neural models are mostly dense, necessitating a different index structure. [12] Furthermore, conducting nearest neighbor matching by considering *all* document representations in a large collection is no longer practical. As a solution, approximate matching is commonly employed in practice. In this approach, queries are executed on vector indices containing these dense representations, utilizing *Approximate Nearest Neighbor (ANN)* matching for efficient retrieval. (Aumüller et al., 2017).
Neural retrieval models are categorized into two approaches: dense and sparse methods. Dense retrieval relies on dense embeddings for document and query representations, feasible only through parallelized dot product computations across multiple GPUs/TPUs and efficient nearest-neighbor search. However, they remain slower than databases with sparse inverted indices and techniques like BM25. Sparse models aim to create a feasible sparse representation similar to traditional models that can take advantage of fast index structures. Dense neural systems are divided into two categories based on the level of interaction between the query and the document, including *term-level representation learning* and *document-level representation learning*. Sparse neural models are categorized into *neural weighting scheme* and *sparse representation learning*.
In the subsequent subsection, we begin with an overview of loss functions and training paradigms for neural models. This is followed by a discussion of ANN methods for efficient retrieval of documents using dense representation. These foundational topics are important for understanding the common practices for training neural IR models and also methods for efficient retrieval. Finally, we provide an overview of the most prominent dense and sparse neural models. We aim to describe at least one model per category, but mainly focus on the model architectures, which are used in Chapter 5 for quantity-aware ranking or as baselines for comparison.

[12] There are ways to create sparse representations from the dense models, which are discussed in the upcoming sections.

2.2.5.1 Loss Functions

Neural ranking models commonly employ either *Negative Log-Likelihood (NLL)* loss for classifying relevant and non-relevant documents or triplet loss functions to learn the margin, which separates the relevant and non-relevant samples.
Classification approaches look at a single document at a time and compute the classification loss based on the relevance of a document to the current query. In this case, the score for each document is independent of the other documents. This is also referred to as NLL loss for a set of binary relevance judgments. Here, the idea is to maximize the probability of relevance for positive samples, given a query. Given D^- as the list of all non-relevant documents for a query. For a query x and a relevant document d^+, the NLL loss is defined in Equation 2.27. Given that computing the normalization term across all documents is time-consuming, *negative sampling* is commonly employed as a workaround to calculate the denominator.

$$L_{NLL}(x, d^+) = -\log \frac{\exp(\tau(x), \tau(d^+))}{\exp(\tau(x), \tau(d^+)) + \sum_{d' \in D^-} \exp(\tau(x), \tau(d'))} \tag{2.27}$$

In negative sampling, we sample a set of non-relevant documents as negatives, denoted by N_x, and compute the denominator based on these samples. In the most basic case, the documents are chosen at random. The reformulated NLL equation is shown in Equation 2.28 (Karpukhin et al., 2020; Qu et al., 2021).

$$L_{NLL}(x, d^+) = -\log \frac{\exp(\tau(x), \tau(d^+))}{\exp(\tau(x), \tau(d^+)) + \sum_{d' \in N_x} \exp(\tau(x), \tau(d'))} \tag{2.28}$$

NLL loss is not dedicated to binary classification and can handle multiple classes. However, most times in relevance classification, the relevance judgments are binary. Another way to maximize the log-likelihood for binary classification is to minimize the *Binary Cross Entropy (BCE)*, which measures the average number of bits needed to represent the outcomes of a binary variable. Given the representations of a query and a document, the probability of their relevance is computed by the sigmoid (σ) of some vector operation ($\odot$), e.g., concatenation or dot product. The BCE loss is shown in Equation 2.29, where g is a linear function and y is the relevance judgment.

$$\begin{aligned} p(d, x) &= \sigma(g(\tau(x) \odot \tau(d))) \\ L_{BCN}(x, d) &:= -y.p(d, x) - (1 - y).(1 - p(d, x)) \end{aligned} \tag{2.29}$$

In triplet loss (Schroff et al., 2015), each query in the training set is paired with both a relevant and a non-relevant sample. To calculate the loss, two forward passes are required: one to compute the representation of the relevant sample and another for the non-relevant one. Triplet loss aims to push

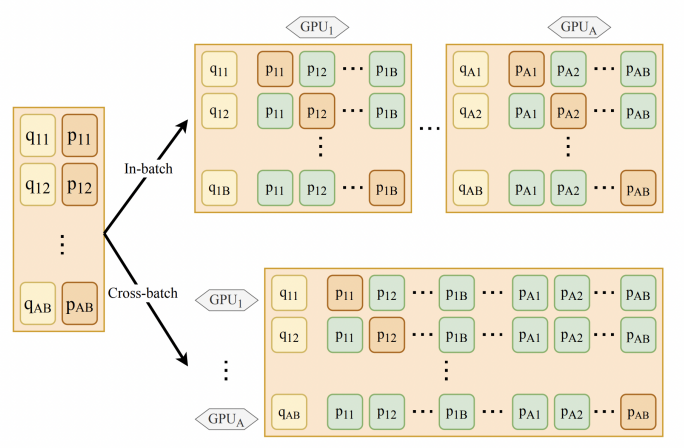


Figure 2.15: The comparison of in-batch negatives and cross-batch negatives when trained on multiple GPUs, where A is the number of GPUs, and B is the number of queries in each batch; Image taken from Qu et al. (2021).

dissimilar pairs farther apart from similar pairs by a predefined margin value. Formally, for a query x and a relevant document d^+ and non-relevant document d^-, triplet loss is defined in Equation 2.30.

$$L_{triplet}(x, d^+, d^-) := max(0, 1 - (sim(\tau(x), \tau(d^+)) - sim(\tau(x), \tau(d^-))) \tag{2.30}$$

Once again the negative sampling strategy plays an important part in convergence. Random samples are easy to distinguish for most models and result in slow convergence and low-quality representations. As a result, different strategies for mining *hard* or *semi-hard negatives* have been proposed. Here, we discuss a few of them. It is important to note that the choice of positive and negative examples significantly impacts the learning process, guiding the model on which features to prioritize in distinguishing relevance.

The major negative sampling methods are divided into three categories (Zhao et al., 2022):

1. *In-batch negatives:* This is a straightforward and efficient way to acquire negative samples, first introduced by Henderson et al. (2017). Instead of mining negatives from the entire collection, for each query, the positive text is paired with the text of other queries in the same batch as negatives. Since neural models are trained on GPUs with limited memory, this technique provides a large number of negatives within the same batch and with minimal memory cost. If there are B queries in a batch and each query has exactly one relevant text, with in-batch negatives there are $B - 1$ negatives in the same batch.

2. *Cross-batch negatives:* This method is suitable for multi-GPU settings, where the examples are reused across different GPUs (Qu et al., 2021). Here the document embedding is computed in a GPU and shared among all other GPUs. Besides the in-batch negatives, all the other documents from the other GPUs are additional negative samples. If we have A number of

GPUs, the number of negatives increases to $A \cdot (B - 1)$. An illustration of the difference between in-batch and cross-batch negative sampling is shown in Figure 2.15. The idea of cross-batch negatives can be extended to a single GPU by accumulating the negatives across multiple mini-batches but it is less common (Gao et al., 2021).

3. *Hard negatives:* These types of negatives are essentially irrelevant text passages that exhibit a high degree of textual similarity to the relevant ones. The inclusion of hard negatives in training increases the discriminating capabilities of retrieval systems (Karpukhin et al., 2020; Qu et al., 2021). Hard negatives are further classified into three categories: *static hard negatives*, *dynamic hard negatives*, and *denoised hard negatives*.

    a) *Static hard negatives:* These negatives are derived from a selector that remains constant throughout the training process. A commonly used selector is BM25, which retrieves lexically similar answers to the queries but does not contain the actual answer. (Karpukhin et al., 2020). Other choices of static hard negatives are sampling from similar contexts or topics or a mix of all approaches. For instance, in the case of a relevant document containing multiple passages, all passages except the relevant one are labeled as negative. Another approach involves employing an efficient dense encoder to identify semantically similar passages (Lu et al., 2021). Li et al. (2022) propose a topic-aware sampling, in which queries are clustered into topics, and the negatives are chosen from the same cluster.

    b) *Dynamic hard negatives:* These types of negative samples come from an adaptive negative selector. ANCE (Xiong et al., 2021) selects hard training negatives globally from the entire corpus, using an asynchronously updated ANN index. The index is refreshed parallel to the training based on the updated model parameters. Consequently, hard negatives are selected from the latest embeddings. This synchronized approach to selection promotes faster convergence. ADORE (Zhan et al., 2021) is another approach, which keeps the text embedding index fixed and utilizes an adaptive query encoder, resulting in retrieving adaptive negative samples.

    c) *Denoised hard negatives:* In this scenario, noisy negative samples are filtered out to enhance quality, specifically targeting false negatives. Noisy negatives often stem from top-ranking retrieval samples generated by static or dynamic systems that closely match the query, but are actually positives, thereby introducing noise into the negative examples. To resolve this issue RocketQA (Qu et al., 2021) proposes a denoised negative selection using a well-trained *cross-encoder* [13] to filter top-ranked retrieval results that are likely to be false negatives. The cross-encoder is one of the most powerful similarity indicators that can be used to refine the selection. Another denoising technique is SimANS (Zhou

[13] Cross-encoder is described in more detail in a later section. It is an encoder-based language model where both passages are fed into the model divided by the `[SEP]` token.

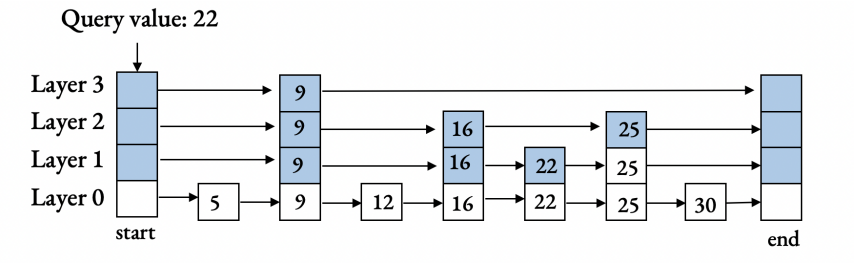


Figure 2.16: Probability skip-list and the search for value 22.

et al., 2022), which selects ambiguous negatives (those that are ranked close to positives based on relevance scores). These ambiguous negatives are considered high-quality negatives as they strike a balance between being neither too easy nor too difficult, potentially mitigating the presence of false negatives.

2.2.5.2 Approximate Nearest Neighbour (ANN)

ANN algorithms accelerate similarity search on index structures for dense vectors by replacing exhaustive comparisons between all document embeddings and a given query with a more computationally efficient approximation. This approximation is achieved through techniques such as vector compression, limiting the search scope, or reducing the required memory. Generally, ANN algorithms are categorized into four major types: *tree-based* (Beis and Lowe, 1997; Bentley, 1975), *hashing-based* (Datar et al., 2004; Indyk and Motwani, 1998), *quantization-based* (Ge et al., 2014; Jégou et al., 2011), and *proximity graph approaches* (Malkov and Yashunin, 2020).
The earliest approaches for ANN used hashing-based techniques, but currently, proximity graphs offer the best performance. Graph-based methods construct an index by preserving neighborhood information for each data point or a set of pivot points. Various greedy heuristics are then applied to these graphs to navigate to the closest points for a given query. Unlike tree-based and hashing models, graph algorithms are metric-agnostic, meaning they can work with a wide range of similarity metrics. Graph models not only outperform other methods but also maintain a local view, making it easier to add new data with minimal distribution changes. Several open-source libraries for ANN search exist, among which *Facebook AI Similarity Search (Faiss)* (Johnson et al., 2021) from Facebook (Meta) and *Space Partition Tree And Graph (SPTAG)* (Chen et al., 2018) from Microsoft Research are well-known. Since the underlying principle of the graph-based methods is the same, we describe *Hierarchical Navigable Small World (HNSW)* (Malkov and Yashunin, 2020), which is the underlying technology of Faiss.

HNSW is a type of proximity graph, where nodes are embeddings that are connected to other nodes based on their proximity. The proximity is commonly measured by Euclidean distance. HNSW is based on two fundamental ideas: *Navigable Small World graphs (NSW)* (Malkov and Yashunin, 2020) and *probability skip-list* (Pugh, 1990).

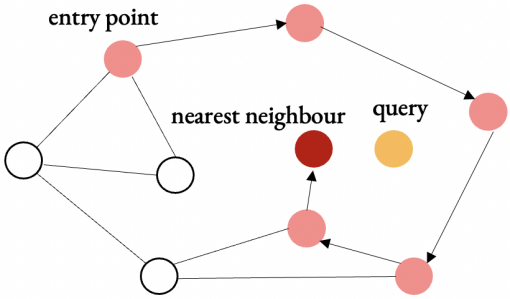


Figure 2.17: An example of a greedy-routing search for the query marked in yellow. The search starts at the entry point and continues until the local minima is reached.

A probability skip-list was introduced in 1990 and is based on building a linked list in several layers. On the first layer, there exist links that skip many intermediate nodes and as the layers go deeper, the number of skips by each link is decreased. An example of such skip-list and search for an example value of 22 is depicted in Figure 2.16, where the search is started at the highest level. Instead of examining all values, the search begins at the top layer, comparing the query value to the values in that layer until a larger value is encountered, overshooting the target. Once a larger value is found, the search moves down to the next layer and continues. This process repeats until the query value is found. The purpose of organizing the list into multiple layers is to minimize the number of steps needed to locate the final value. In Figure 2.16, we only have to traverse the nodes in blue to find the final value, and intermediate values of 5 and 12 are skipped.

Vector search with navigable small world graph was developed over the course of several papers (Malkov et al., 2014; Ponomarenko et al., 2011; Malkov et al., 2012). The main idea is to reduce search time to logarithmic complexity by constructing a graph with both long and short-range links based on node proximity. Each node is connected to its neighbors with an edge, these neighbors are referred to as friends, and for all nodes, a friends list containing all neighbors is kept. Searching is conducted using *greedy-routing search*, where the algorithm navigates the graph by selecting the friend closest to the query node as the next node to traverse. Each graph has a predefined entry point where the search begins. At each step, the closest node from the current friend list is chosen as the next destination. This process continues until no node closer than the current one can be found, indicating a local minimum. Figure 2.17 shows a hypothetical example of greedy traversing at a high level. The search starts at the entry point and the traverse is continued until the local minimum is reached, and the current node on the traverse path is returned as the nearest neighbor (dark red node). Nodes with few friends (low degree) are more likely to get stuck in a local minimum. In contrast, high-degree nodes offer more options and are less likely to be trapped in local minima. High-degree nodes are also more likely to have long-range edges that extend beyond the local cluster.
Routing within the graph involves two phases. The *zoom-out* phase navigates through low-degree

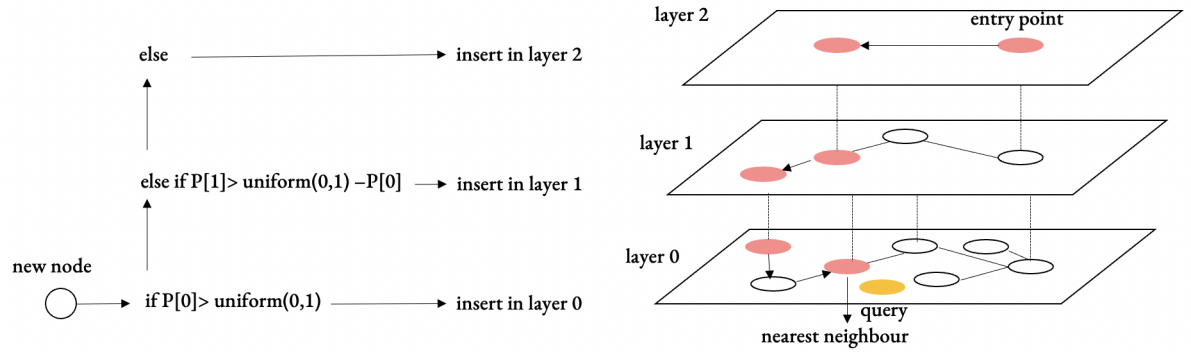


Figure 2.18: An example of node insertion (on the left) and search (on the right) in NHSW.

nodes, while the *zoom-in* phase traverses higher-degree nodes. Consequently, hitting a local minimum is more likely during the initial zoom-out phase. To reduce the likelihood of early termination, higher-degree nodes are necessary, though this increases computational cost. An alternative approach, used by HNSW, is to start the search with high-degree nodes (zoom-in first).
HNSW brings together the probability skip list and NSW. The major difference to a skip list is that the linked list is replaced with proximity graphs. Edges are like skip connections that are separated across different layers, where the top layer contains long-range edges (long skips) and nodes with higher degrees. The search starts at the top layer with high-degree nodes, reducing the chance of reaching a local minimum. On each layer, traversal among the nodes is similar to NSW, moving greedily to the nearest node until a local minimum is found. However, unlike NSW, the search does not stop at the local minimum but instead descends to the next lower layer. This process is repeated until the local minimum is reached at the bottom layer (layer 0). An illustration of the search on an example graph is depicted in Figure 2.18, where the arrows show the direction of the greedy algorithm from the entry point to the nearest neighbor of the query.

Figure 2.18 also depicts the process of graph construction and the insertion of new nodes. Nodes are iteratively inserted one by one during graph construction. For each new element, an integer representing the maximum layer is randomly selected based on an exponentially decaying probability distribution P. The likelihood of insertion decreases exponentially with each ascending level in the hierarchy. The probability of insertion is highest at the lower layers, particularly at layer 0, while only a few nodes make it to the top layer. Layer 0 contains all the nodes in the graph. If a node is inserted into a top layer, it will also be included in all the lower layers. For example, nodes at layer 2 will also be present in layers 1 and 0. When a node is added to the selected layer, the graph is traversed greedily from the top layer to find ef closest neighbors to the new element. This search continues layer by layer, using the closest neighbors found in the previous layer as starting points for the next layer. The parameter ef is the approximate number of neighbors to be maintained for each node. The list of closest friends is updated on each insertion by looking at the neighborhood of the closest previously non-evaluated element in the list. For a detailed algorithm on search and graph construction, refer

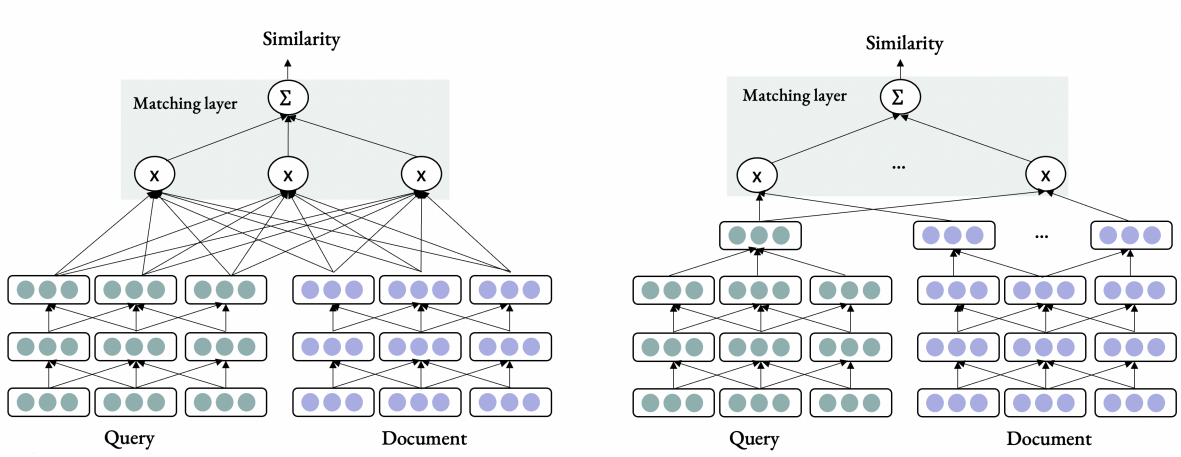


Figure 2.19: Dense retrieval models based on interaction level. The left side is the term-level representation and the right side is document-level representation. Figure inspired from (Guo et al., 2022)

to the paper by Malkov et al. (2014).

With advances in neural IR and their superior performance in comparision to the classical methods, and open-source search engines such as the ones mentioned in Section 2.1.3 are increasingly adopting vector databases and techniques like HNSW. Moreover, databases such as Redis [14] and Weaviate [15] offer extremely efficient vector search index structures, which work with a variety of embedding methods. However, these open-source search engines and databases mainly focus on the most basic query and document representation, where a single dense vector represents the query and another represents the document. In the upcoming sections, we explore methods where embeddings are either sparse or where a single query and document are represented by multiple vectors. Despite the superior performance of such models, their support in open-source software remains limited.

Now that we covered the training paradigm, transformer-based language models, and ANN search, we can delve into neural retrieval models. In the upcoming sections, we cover various model architectures for dense and sparse retrieval, most of which are built on top of pre-trained encoder-based language models. During fine-tuning on positive and negative samples, these models adapt their internal representation to capture a notation of relevance between a query and a document.

2.2.6 Dense Neural Models

The essence of dense neural retrieval is ranking based on semantic interaction between the dense representations of a query and documents. Based on different interaction levels, dense models are divided into *document-level representation* and *term-level representation*.
Term-level representations show more fine-grained interactions among terms of query and docu-

[14] https://redis.io/(lastaccessed:02.05.2024)
[15] https://weaviate.io/(lastaccessed:02.05.2024)

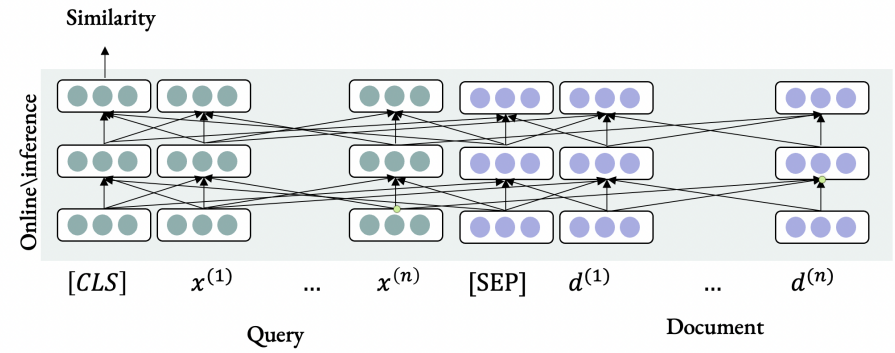


Figure 2.20: Cross-encoder architecture, where the [CLS] embedding is used to predict the relevance/similarity. Both query and document representations are computed online (during inference).

ment compared to document-level, where all the terms are combined into a single representation. Figure 2.19 shows the difference in the architecture of these two types. In the following subsection, each variation is described in more detail along with respective prominent models. Since the query representation might originate from a different network than the document representation, we will refer to the query representation as $\tau'(x)$ and the document representation as $\tau(d)$.

2.2.6.1 Term-level Representation

Term-level models learn a fine-grained (token-level) representation for a term in a query and a document. As shown in Figure 2.19 on the left side, the matching function calculates the interaction between the embedding of the terms in the query and document.

Cross-encoder: The most comprehensive yet most computationally expensive form of term-based interaction is a cross-encoder. A visualization of the cross-encoder is shown in Figure 2.20. In this case, both query and document are fed into a pre-trained language model, e.g., BERT, separated by the [SEP] token. Due to the self-attention and cross-attention in the transformer architecture the semantic interaction *within* all the terms as well as *across* terms in query and document are captured. Then either the [CLS] token (Nogueira and Cho, 2019; Qiao et al., 2019) or a transformation of the token embeddings, e.g., average or max (Xiong et al., 2017) is used to predict the relevance. Although the cross-encoder is a powerful ranker, its practical use is infeasible. Ranking a single query requires computing the interaction between the query and all documents in the corpus in real time, making these models an expensive choice even as a re-ranker. To address this, recent models aim to delay the interaction computation and make it more efficient while retaining the benefits of term-level interactions. We discuss a few of the relevant works and refer the reader to surveys of neural retrieval models for more model variations, e.g., surveys by Guo et al. (2022) and Zhao et al. (2022).

ColBERT: The *Contextualized Late Interaction over BERT (ColBERT)* model (Khattab and Zaharia, 2020) proposes *late-interaction* to balance between the quality and cost of term-level inter-

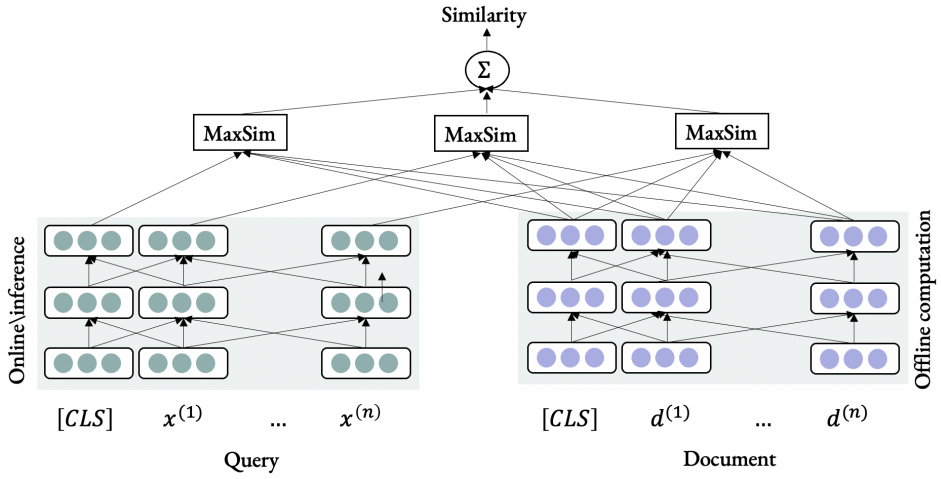


Figure 2.21: ColBERT model and the late-interaction mechanism. The document representations are created and indexed offline. During inference time only the query representation is computed and compared against the index for efficient retrieval.

actions. Figure 2.21 depicts the architecture of ColBERT, where a query encoder and document encoder compute $\tau^{'}(x)$ and $\tau(d)$ with a bag of fixed-size token embeddings. As a result, the query and document representations are comprised of the embeddings of all their tokens. Each token embedding is contextualized based on the surrounding terms. The relevance score between these fine-grained embeddings is computed using a late-interaction mechanism, specifically by summing the maximum similarity (MaxSim) values. MaxSim operator returns the similarity score for the most similar token between query and document. Query and document encoders are a BERT model with shared parameters, but there are some differences in how they are encoded.

Query Encoder: a special token is prepended to the input to denote if it is a query or a document, `[Q]` for the query, and `[D]` for the document. $\tau^{'}(x)$ and $\tau(d)$ are WordPiece representations, due to the wordpiece tokenization of the BERT model (described in Section 2.2.4).
BERT model has a limited sequence length (number of tokens that are allowed as input), and if the query has fewer tokens than the defined limit, it is padded with the special `[Mask]` token. The authors claim that the mask tokens perform an implicit *query augmentation*, by allowing BERT to predict the mask token based on the query context. The BERT representation for each token, including the masked ones, is passed through a linear layer with no activation to reduce the size of the final embeddings and reduce the space footprint.

Document Encoder: Unlike the query encoder, no mask is appended to the documents, and short documents are padded normally. After the final linear layer, token embeddings corresponding to punctuation are removed from the final representation to reduce the memory footprint. Both query and document representation are normalized to unit length.

Using the query and document encoders $\tau^{'}(x)$ and $\tau(d)$ for $x = x^{(1)}...x^{(n)}$ and $d = d^{(1)}...d^{(n)}$ are computed using Equation 5.14, where # denotes the mask tokens. Note that the masking is only applied to the query encoding. The CNN layer is a 1-D convolution to down-project the embeddings and the Filter function removes the punctuation embeddings after document encoding.

$$\begin{aligned}\tau^{'}(x) &:= \text{Normalize}(\text{CNN}(\text{BERT}([\text{Q}]x^{(1)}...x^{(n)\#...\#})))\\ \tau(d) &:= \text{Filter}(\text{Normalize}(\text{CNN}(\text{BERT}([\text{D}]d^{(1)}...d^{(n)}))))\end{aligned} \tag{2.31}$$

Given $\tau^{'}(x)$ and $\tau(d)$, the similarity is estimated using late interaction with cosine, implemented as the dot product of the normalized vectors. More formally the MaxSim operator for query x and document d is shown in Equation 2.32 as the sum of the dot products of all individual query and document token embeddings. This approach measures similarity by selecting the most similar tokens from the query and the document. ColBERT is trained using cross-entropy loss, with positive and negative samples to optimize this similarity measure.

$$sim(x,d) = \sum_{i \in ||\tau^{'}(x)||} max_{j \in ||\tau(d)||} \tau^{'}(x)_i . \tau(d)_j \tag{2.32}$$

ColBERT can be used both for first-stage (end-to-end) retrieval and also as a re-ranker. For end-to-end retrieval, the pruning-friendly nature of the MaxSim operator is highly beneficial. Document embeddings are precomputed offline and stored in a fast vector-similarity index, such as Faiss. The MaxSim operator, in conjunction with Faiss, is used to retrieve the nearest neighbors to a query based on the token embeddings of both the query and the documents. This initial list of nearest neighbors is then refined through re-ranking. For re-ranking, the dot product between the query token embeddings and the document representations is computed to measure similarity. A max-pooling layer on top selects the maximum similarity scores. ColBERT efficiently returns query results on a large corpus within a short time frame, but it still lags behind classical methods like BM25.

In Chapter 5, where we propose extensions to neural models for quantity understanding, ColBERT is chosen as a representative of dense neural IR. This choice is driven by ColBERT's balance of performance and efficiency, as well as its use of term-level interactions modeled through the MaxSim operator on term embeddings. We argue that maintaining term-level interactions is crucial for answering quantity-centric queries, where the relationship between terms and quantities is significant. Such fine-grained connections are lost in document-level representations, where a single vector represents the entire document and query.

Multiple extensions of ColBERT have been proposed throughout the years. ColBERTv2 (Santhanam et al., 2022) boosts the performance by a combination of distillation from a cross-encoder and hard-negative mining. Additionally, it uses a residual compression mechanism to reduce the

space footprint. ColBERTer (Hofstätter et al., 2022) proposes an extension to increase the level of interpretability and reduce the storage and latency cost. To enhance interpretability, the Bag of Whole-Words approach is introduced. This method aggregates all sub-word token representations within a unique whole word into a single representation. This not only improves interpretability but also reduces the number of vectors that need to be stored per document.

2.2.6.2 Document-level Representation

The document-level representation models learn one or more global representations for query and document. The similarity between the global representation shows the final relevance, which is often computed with a dot product or cosine similarity. Although earlier models obtained query and document embeddings by aggregating their corresponding word embeddings with heuristic functions (Clinchant and Perronnin, 2013; Gillick et al., 2018), recent models use transformer models.

Dense Passage Retrieval (DPR): This is the first model in this category to combine language models for global document and query representations by using a BERT-dual encoder. The bi-encoder architecture introduced in DPR has since been used in a variety of other models for efficient end-to-end dense retrieval. The general bi-encoder architecture is shown in Figure 2.22. In DPR, a dense encoder precomputes dense representations, $\tau(d)$, for documents offline. During runtime, a separate encoder maps the query to a dense vector, $\tau^{'}(x)$. The similarity between a query and each document is then estimated using the dot product, as shown in Equation 2.33. Note that in DPR, unlike the term-level models, there is a global vector abstracting the features of the query and document and the token-wise interactions are ignored.

$$sim(x, d) := \tau^{'}(x)_{cls} \cdot \tau(d)_{cls} \tag{2.33}$$

DPR is trained using NLL with a positive document and negative sampling. The authors experiment with different negative sampling strategies and finally settle on a combination of in-batch negatives and hard negatives from BM25.

Document embeddings are stored in Faiss, for efficient k-nearest neighbor retrieval during inference. DPR can serve either as a re-ranker or can be used for end-to-end retrieval. In the re-ranking scenario, DPR is commonly paired with a lexical matcher like BM25. In end-to-end retrieval, a large number of nearest neighbors are retrieved using Faiss, and fine-grained matching is conducted using Equation 2.33. DPR was one of the first models that introduced the idea of efficient retrieval with large language models, hence its popularity in literature.

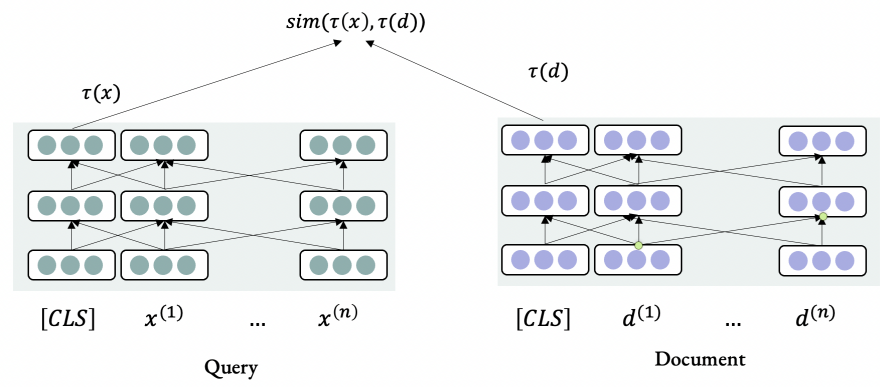


Figure 2.22: Bi-encoder architecture for retrieval, based on the DPR model, where the representation of query and document is the [CLS] embedding of the BERT encoder.

An improvement on DPR is proposed through RocketQA (Qu et al., 2021), which uses a bi-encoder but shows that the optimization of the training procedure has a great impact on the final result. Specifically using denoised hard negatives and data augmentation with the help of a cross-encoder can enhance results significantly.

Knowledge distillation: In general, cross-encoders are used in a variety of ways to enhance the performance of retrieval systems. Knowledge distillation is one way to take advantage of the knowledge learned by these expensive models. Knowledge distillation refers to the process of transferring knowledge from a more capable and usually larger model (called teacher) to a smaller and less capable model (called student). The teacher is usually a well-trained cross-encoder who is capable of capturing the semantics and interactions between query and document (Tahami et al., 2020). However, in some cases, models like ColBERT are also used as the teacher (Lin et al., 2020b). Given that bi-encoders are comparatively more cost-effective models, they are predominantly utilized by students. Knowledge distillation is usually done in two ways: *hard-label distillation* and *soft-label distillation*.

Hard-label distillation: This method uses the output of the teacher as ground truth for training the student (Qu et al., 2021). Given that the teacher's predictions may include noisy output, robust thresholding techniques are employed to filter out low-confidence scores. In this approach, a set of query and document pairs with unknown judgments are tagged into relevant and non-relevant classes using a cross-encoder. The newly tagged set is added to the training data of the student to enhance its performance.

Soft-label distillation: This method aims to approximate the output distribution of the teacher using the student network, by adjusting the loss function to learn the output probability of the teacher. If $sim_t(x, d)$ represents the relevance score from the teacher and $sim_s(x, d)$ from the student, several distillation loss functions include (Izacard and Grave, 2021; Zhao et al., 2022):

1. *MSE loss*: In this case, the loss function minimizes the difference between the relevance of the teacher and the student using mean square, as shown in Equation 2.34.

$$L_{MSE} := \frac{1}{2} \sum_{x \in X} \sum_{d \in D} (sim_t(x, d) - sim_s(x, d))^2 \tag{2.34}$$

2. *KL-divergence loss*: This function first normalizes the relevance scores of candidate documents into probability distributions by queries, denoted by, $\tilde{sim}_t(x, d)$ and $\tilde{sim}_s(x, d)$, and reduces their KL-divergences as in Equation 2.35. KL-divergence measures the distance between two distributions and by minimizing it, the student learns to make similar predictions as the teacher.

$$L_{KL} := - \sum_{x \in X, d \in D} \tilde{sim}_s(x, d)(\log \tilde{sim}_s(x, d) - \log \tilde{sim}_t(x, d)) \tag{2.35}$$

3. *Margin-MSE loss*: This loss reduces the margin difference between the positive and negative predictions of the teacher and the student, as shown in Equations 2.36.

$$L_{MMSE} := \text{MSE}((sim_t(x, d^+) - sim_s(x, d^+), (sim_t(x, d^-) - sim_s(x, d^-)) \tag{2.36}$$

2.2.7 Sparse Neural Models

In the sparse retrieval scheme, each document and query is represented with sparse vectors, where only a small number of dimensions are active. The difference to the classical models is that the representations come from a neural network, where the semantics and relationships are learned during training. The sparsity is either created from a dense representation directly or a neural model is used to predict token importance in documents, which replaces the term frequencies in classical models. A great advantage of sparse models is that they can be easily integrated into existing inverted index structures for efficient retrieval. Sparse retrieval models are categorized into two classes: *neural weighting scheme*, refers to employing the neural models to improve term weighing, where the classical inverted indices are used for retrieval. On the other hand, *sparse representation learning* directly learns queries and documents in a sparse space using neural networks.

2.2.7.1 Neural Weighting Scheme

One of the major components in ranking for classical models is the term weights stored in an index structure, which indicate their importance. [16] Neural weighting models, however, either employ a neural network to learn better term importance based on semantics rather than heuristics, or they keep the weights intact but augment documents with additional terms before indexing, using neural

[16] Frequencies are treated as weights in the classical models.

networks for semantic expansion (Guo et al., 2022). Based on these approaches, neural weighting schemes are divided into *term re-weighting*, *expansion*, and *expansion + term re-weighting*.

Term re-weighting: These models focus on estimating term importance in queries and documents using neural networks. *Deep Contextualized Term Weighting (DeepCT)* (Dai and Callan, 2019) is an example of such models, which primarily focuses on term re-weighting through contextualized representations with BERT. To estimate the importance of tokens in the text, DeepCT first encodes the passages and queries using BERT. The output of the BERT encoder is a contextualized feature vector that captures the syntactic and semantic meaning of tokens. This vector is then used as input for a linear regression model to predict the importance score of each word within the context of the entire text. The linear regression layer is a simple feed-forward network. Since in the BERT model sub-words are used as tokens, in order to compute the weight for a word, only the weight of the first sub-word token is used.
More formally, importance is predicted using a linear combination of BERT token embedding ($\tau(d)^t$) as shown in Equation 2.37, where y_d^t is the predicted weight for a token t in a document d and W, b is the weight and biases, respectively.

$$y_d^t := W^T \tau(d)^t + b \tag{2.37}$$

DeepCT is trained using a per token regression task with MSE, to predict the correct weight for each token. However, since the true term weights are unknown, the authors estimate these values using *query term recall*. Equation 2.38 shows the computation for query term recall, which is based on the assumption that the search queries reflect the topic of a document. Words appearing in queries relevant to a document are considered more important than other words within the document. In this context, Q_d is the set of queries for which document d is relevant and $Q_{d,t}$ is the subset of those queries containing token t. The importance score, or query term recall, is calculated as the ratio of the number of queries containing the term to the total number of relevant queries for the document.

$$QTR(t,d) := \frac{|Q_{d,t}|}{|Q_d|} \tag{2.38}$$

After training, the predicted term weights replace the term frequencies in an inverted index and can be combined with widely used retrieval models like BM25. Hence, the main computation overload of DeepCT is prior to index creation.
To compute the term weighting for queries, the authors propose a similar approach. In this case, the term importance is estimated by *term recall*, as shown in Equation 2.39. D_d is a set of documents relevant to the query and $D_{d,t}$ is the subset of those documents containing token t.

$$TR(t,d) := \frac{|D_{d,t}|}{|D_d|} \tag{2.39}$$

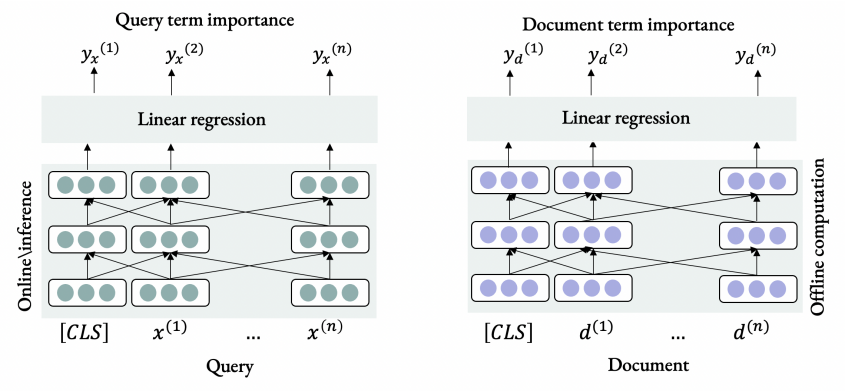


Figure 2.23: Architecture overview of DeepCT model with query and document encoder. Term importance scores for query and document come from query term recall and term recall respectively.

Term recall is based on the assumption that a token is more important if it is mentioned in relevant documents. The overall architecture of DeepCT is shown in Figure 2.23.

Expansion: One technique to improve classical retrieval models is to expand documents with relevant terms. By enriching documents with expansion terms, cases of vocabulary miss-match decrease. Doc2Query (Nogueira et al., 2019b) utilizes a sequence-to-sequence transformer for such document expansions. For each document, the task is to predict a set of queries that are relevant. The model is trained using query and relevant document pairs. An encoder computes a representation of a document, and a decoder predicts the relevant query based on the document representation. Once the model is trained, several queries are generated for each document and appended to the existing document terms as an expansion. The expanded documents are then indexed, and the final ranking is performed using classical methods such as BM25. A further extension of Doc2Query is to use the T5 model (Raffel et al., 2020) instead of BERT for better effectiveness (Nogueira et al., 2019a). T5 is an encoder-decoder architecture and is better suited for conditional text generation than BERT.

Expansion + term re-weighting: These models perform both expansion and estimation of term importance and are the type that we use in Chapter 5. One relatively simple yet effective model is SPLADE (Formal et al., 2021b). SPLADE is inspired by the SparTerm model (Bai et al., 2020), which directly learns term-based sparse representation for the entire vocabulary space. Figure 2.24 depicts the general architecture of SparTerm, consisting of an *importance predictor* (F) and a *gating controller* (G). The final sparse representation for a document is a combination of the outputs of the importance predictor and gating controller ($\tau(d) = F(d) \odot G(d)$), where $\odot$ is the element-wise multiplication of the two vectors.
The importance predictor F generates a dense vector representing the semantic importance of each term in the vocabulary, thereby achieving both term weighting and expansion. Concurrently, the gating mechanism filters out less important terms, ensuring sparsity. To calculate the importance,

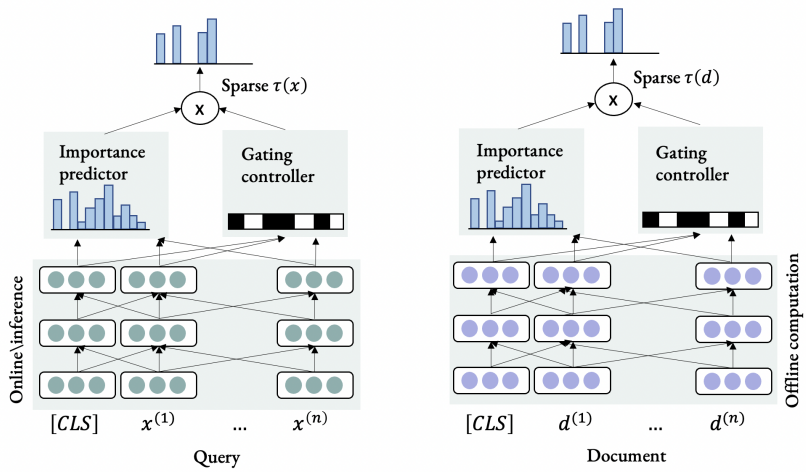


Figure 2.24: SparTerm model uses a gating and importance module on top of BERT representation to filter terms and compute importance based on the entire context.

the contextualized embeddings from BERT undergo a linear transformation with GELU activation and layer normalization to predict token-wise importance scores. The final document importance distribution (I) is the summation of all token-wise importance distributions, as shown in Equation 2.40. ReLU activation is used to ensure the non-negativity of the output. Here, W and b represent the weights and biases, respectively, and $I^{(i)}$ is a distribution over the vocabulary, equivalent to the MLM prediction. This equivalence allows for initialization from a pre-trained BERT.

$$
\begin{aligned}
I^{(i)} &:= \text{Normalize}(\text{GELU}(\text{BERT}(d^{(i)})))W^T + b \\
I &:= \sum_{i=0}^{n} \text{ReLU}(I^{(i)})
\end{aligned}
\tag{2.40}
$$

More precisely, the importance predictor predicts term importance across the entire BERT WordPiece vocabulary based on the logits of the masked language model. $I^{(i)}$ is therefore a vector with the size of the vocabulary where each element denotes the importance of the ith token of the input to that token in the vocabulary. By summing up across the token axis, we get the importance of all the terms in the vocabulary based on the current input tokens.

The gating G controls which terms appear in the final representation with binary gating. SparTerm proposes two types of gating mechanisms: *literal-only gating* and *expansion-enhanced gating*.

Literal-only gating is similar to a bag-of-words representation of a document, where only existing terms in the original document are activated and no expansion is performed. On the other hand, expansion-enhanced gating activates terms that might bridge the lexical mismatch gap. Similar to importance prediction, a document-wise dense term gating distribution G is computed to quantify the probability of each term participating in the final sparse representation. To ensure sparsity, a binary function is applied on top of outputs as a binary activation function, which outputs only 0 or 1 ($G' = \text{Binarize}(G)$). The final expansion-enhanced gating vector G_{le} is computed in Equation 2.41,

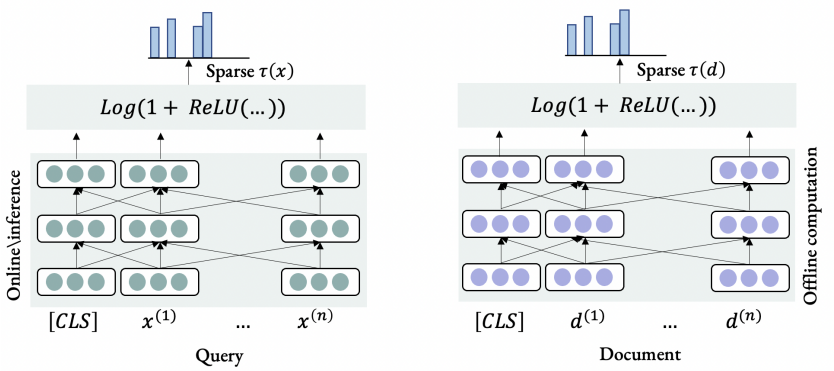


Figure 2.25: SPLADE combines the gating and importance mechanism in a single log saturation layer.

where a bitwise negation vector is applied to ensure only the terms not present in the document are being expanded. The gating vector is the same size as the importance vector and spans across the entire vocabulary. However, unlike the importance vector, the gating vector contains zeros and ones to mask out the irrelevant tokens for the representation.

$$\begin{aligned} G_e &:= G^{'} \odot (\neg \text{BoW}(d)) \\ G_{le} &:= G_e + \text{BoW}(d) \end{aligned} \tag{2.41}$$

The final representation $\tau(d)$ for each document comes from the element-wise multiplication of binary masking and the importance vector for each term. SparTerm is trained by minimizing a token-wise NLL loss for ranking and BCE loss for the gating mechanisms. Separating the importance of predictors and gating mechanism is rather complex and hard to train.

The SPLADE model aims to simplify the importance and gating module into a single one by introducing an importance estimation that already imposes sparsity without the need for an extra gating mechanism. Both modules are combined under a log-saturation layer, which is added on top of BERTs masked language modeling distributions. Equation 2.42 states the log-saturation effect that prevents some terms from dominating and adds sparsity.

$$\tau(d) := \log(1 + \text{ReLU}(\text{BERT}((d))) \tag{2.42}$$

The log saturation trick, similar to $\log(tf)$ in the classical models, manages to add some degree of sparsity by squashing large values. Moreover, the ReLU activation filters out negative values. The difference between the SPLADE architecture and SparTerm is shown in Figure 2.25, where the entire importance and gating module is replaced by a single log-saturation layer. $\tau(d) \in \mathbb{R}^{|V|}$ is a vector of the size of the vocabulary that contains zeros in many places and word importance scores in the rest. The concept involves using log-saturation implicit expansion to enhance documents by

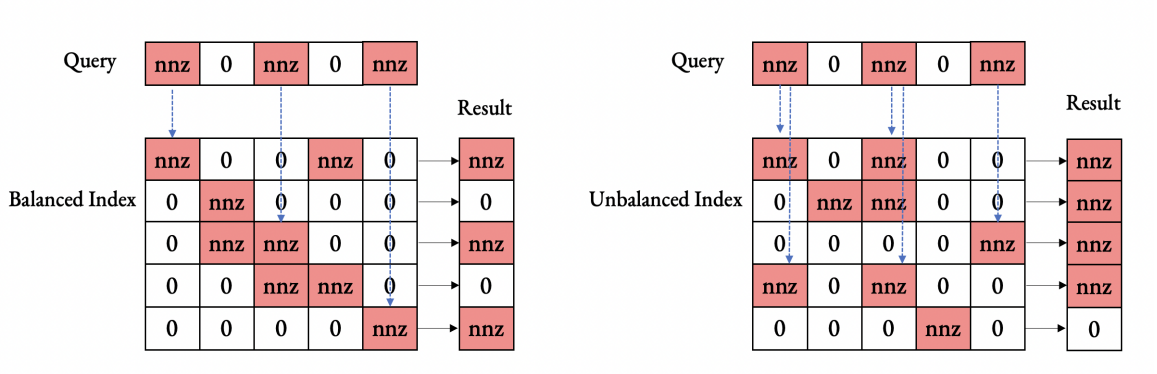


Figure 2.26: Comparison of balanced and unbalanced index with equal sparsity (number of non-zero elements). The distribution of the non-zero elements defines the number interaction.

adding terms while filtering out unimportant ones with low scores. This is achieved through ReLU activation and further squashing by the log function. The resulting sparse vector is then indexed using an efficient structure suitable for classical methods. New terms for a document are selected from the active dimensions of $\tau(d)$, with term weights corresponding to the values of these dimensions. Notably, these new terms might not include the original terms present in the document.

Unlike SparTerm, which requires two distinct loss functions for predictor and gating, SPLADE is trained end-to-end using NLL loss. For negative sampling, the authors utilized the in-batch negatives. Although the log-saturation layer introduces some form of sparsity, in practice still the number of active dimensions are high. To this end, the authors use the *Floating-Point Operations Per Second (FLOPS)* regularizer (Paria et al., 2020), which is a smooth relaxation of the average number of floating-point operations necessary to compute the score of a document. The FLOPS regularizer directly relates to the retrieval time of a document and is more suitable than the l_1 norm.
The primary concept behind FLOPS regularization is to decrease the number of interactions between the query and the document, unlike the L_1 norm which solely focuses on reducing the number of non-zero values. If $\tau(x)^{'}$ is the query representation and $\tau(d)$ is the document representation. Then the l_1 norm is the sum of the elements in $\tau(x)^{'}$ and $\tau(d)$, e.g., $L_{1_x} := \sum \tau(x)^{'(i)}$. This pushes certain values inside the query representation to be zero but it does not define *which* tokens should have zero weight. As a result, this type of sparsity might not result in a speed-up. Figure 2.26 describes this idea better. In this example, an unbalanced inverted index leads to four operations because, despite the documents being sparse, four of them have overlapping terms with the query. In contrast, a balanced index distributes the zeros such that two of the documents are excluded from the final interaction computations, resulting in a speedup. Therefore, the ultimate speed improvement due to sparsity depends not only on the number of non-zero terms but also on their distribution within the index.
In FLOPS regularization, the process is applied to each column of the document embedding matrix (W_d) rather than to each row. This means the regularization focuses on reducing the number of

documents associated with a particular token to only those documents where the token holds significant importance. To estimate the interactions, FLOPS estimates the average number of floating point operations needed to compute the dot product between two representation vectors (query and document) by a smooth function. FLOPS loss is computed as the sum across the terms in the vocabulary of the squared probability of that a term having a non-zero weight in a document (f_j). The computation of FLOPS loss for a single document is shown in Equation 2.43, where j is the index of the tokens in the vocabulary and i is the index of documents in the corpus. During training, access to the entire document corpus is not feasible. Instead, a batch of N documents is sampled to serve as a representative of the entire distribution for computing the regularizer.

$$\begin{aligned} \bar{f}_j &:= \frac{1}{N} \sum_{i=0}^{N} \tau(d)_i^j \\ l_{\text{FLOPS}} &:= \sum_{j \in |V|} \bar{f}_j^2 = \sum_{j \in |V|} (\frac{1}{N} \sum_{i=0}^{N} \tau(d)_i^j)^2 \end{aligned} \tag{2.43}$$

The final loss is the combination of the ranking loss and the regularization on query and document as shown in Equation 2.44, where λ_q and λ_d are the regularization weights.

$$L = L_{ranking} + \lambda_q l^q_{\text{FLOPS}} + \lambda_d l^d_{\text{FLOPS}} \tag{2.44}$$

In Chapter 5, we build upon the SPLADE model for sparse quantity-aware retrieval. We chose SPLADE because it efficiently incorporates both term expansions and re-weighting, combining the best of both approaches. Additionally, one of our quantity-aware methods relies on a quantity-index structure that can be easily integrated into the SPLADE index.

There are two extensions proposed to the SPLADE model. SPLADE V2 proposed several improvements to the base model in terms of effectiveness and efficiency. Query expansion is removed as it is deemed not useful, and knowledge distillation from a cross-encoder is introduced to improve performance (Formal et al., 2021a). SPLADE++ further studies the effects of training improvements like distillation and hard negative mining as well as the choice of language model for base encoders. The authors also investigate the model's performance in terms of zero-shot evaluation.

2.2.7.2 Sparse Representation Learning

These models focus on building sparse representations for queries and documents that capture the semantic meaning of the input text. Inverted indices are used on top of such representations for efficient retrieval, where the units inside the index correspond to *latent terms* instead of actual tokens (Guo et al., 2022). Works in this category are rather limited and since these models are not used in the remainder of the thesis, we only mention the most prominent one.
Learning sparse representations dates back to semantic hashing (Salakhutdinov and Hinton, 2009),

which uses a multi-layer autoencoder for learning representations of documents. Yet, it does not capture the relevance relationship between queries and documents and it is not suitable for retrieval. In 2021, Jang et al. (2021) proposed the UHD-BERT to create high dimensional and sparse embeddings for query and document using BERT. UHD-BERT comprises a query encoder, a document encoder, and a scoring function. The encoder is a BERT model that has a sparsity module on top to create sparse embeddings into buckets. The learned representations are bucketed into multiple parts that represent different aspects of the document or query.

In this chapter, we have explored the major classical and frequency-based retrieval systems. Furthermore, we introduced two families of neural retrieval models: dense and sparse. We laid the groundwork with an exploration of language modeling and transformers to facilitate a deeper understanding of the underlying principles of neural architectures. While none of the models mentioned specifically focus on quantity-centric retrieval or introduce a dedicated architecture and training paradigm for quantities in text, comprehending their internals equips us with the knowledge to incorporate quantity understanding, as discussed in Chapter 5.

3 Related Work on Quantities

> "If I have seen further it is by standing on the shoulders of Giants."
>
> *Issac Newton*

This chapter covers the related work regarding quantity understanding, retrieval, extraction, and question answering. The chapter starts by looking at quantity extraction techniques since many other systems fundamentally rely upon a precise definition and extraction of quantities for their downstream application. By studying prior work in this area, we observe that various systems prioritize different capabilities based on their intended downstream applications, and a unified definition of quantities is lacking.

Section 3.2.1 covers quantity-aware retrieval models and numerical indices and discusses the early work in the area. We point out the differences in the notion of quantity queries and output representations throughout different works in the literature and how they relate to the quantity-aware models in this thesis. Section 3.3 gives an overview of the work in the language modeling domain and numerical embeddings, which aim to create contextual quantity representations. Here, the shortcomings of current embeddings in capturing quantitative values are pointed out, and related work that aims to solve these issues is discussed. Numerical reasoning and question answering systems are discussed next in Section 3.4. These systems are also closely related to quantity representations but mainly focus on optimal architectures or training strategies to inject numeracy and reasoning into language models. Finally, numerical relation extraction is discussed, which aims to detect the relationship between quantities and other concepts in a text. Throughout all upcoming sections, the focus is on describing the methods, contributions, and core findings regarding quantity understanding. The complex mathematical formulation of optimization functions or long tables of evaluation results is, therefore, avoided for a more coherent presentation.

3.1 Quantity Extraction

A building block of most quantity retrieval systems and indices mentioned in the upcoming sections is the correct extraction of quantities. In this section, the related work and advancements in this area are discussed. Extraction of quantities in text goes beyond the identification of mere digits

and is a tedious task, involving standardization and unit detection. Nevertheless, quantity extraction is rarely directly studied in the literature and is often experimented with as part of a larger system. Often the downstream task defines what constitutes a quantity, causing discrepancies between system definitions. This section discusses standalone packages, which are systems specifically designed for quantity extractions, and dependent systems, which are sets of rules or parsers created for numerical data acquisition of a larger system.

Table 3.1 shows a list of dependent and standalone extractors with information regarding various quantity definitions, a summary of their extraction technique, and the type of suitable data. Although most systems consider a value and unit pair as quantity, the representation is mainly dependent on the downstream task or application domain. For example, event-centric methods add a time dimension to the quantities to explore entities through time. The numbering of methods is added to the table rows to easily identify the respective work in the text.

Table 3.1: List of quantity extractors with information regarding quantity definition, extraction technique, focus, and whether they have a code repository available or not.

Model	Quantity Definition	Method	Focus	Code
1. Roy et al. (2015)	(value, unit, change)	semi-CRF and bank of classifiers, set of rules	general	✓
2. Quantulum3	(value, unit)	rules, GloVe-based classifier, dictionary of units	scientific units	✓
3. Huang et al. (2017)	(value, unit)	regular expressions, rules, dictionary of units	5 quantity types	✓
4. Foppiano et al. (2019)	(value, unit)	3 CRF models (unit, value, quantity)	patent documents	✓
5. Banerjee et al. (2009)	(value, unit)	150 rules	10 quantity types	✗
6. Maiya et al. (2015)	(sign, value, unit, error, scientific value)	regular expressions and syntactic patterns	scientific units	✗
7. Alonso and Sellam (2018)	(value, unit, timestamp, property)	rules based on language grammar	event-centric	✗
8. Ning et al. (2022)	(value)	BERT-based span detector	event-centric	✗
9. Ibrahim et al. (2019)	(value, unit)	random forest, random walk	tables	✗
10. Sarawagi and Chakrabarti (2014)	(value, unit)	probabilistic context-free grammar	tables	✗

In Chapter 4, we propose a new framework for quantity extraction from text and provide a more in-depth evaluation of the capabilities of the publicly available standalone extractors. Error analysis on standalone models reveals that domain-specific models, such as Recognizers-Text (Maiya et al., 2015; Huang et al., 2017), often overlook a considerable number of quantities in their extractions. On the other hand, more generalized models like those discussed in (Roy et al., 2015) lack proper normalization and standardization, necessitating extensive post-processing for practical use. However, even with such post-processing, their performance remains poor. [1]

3.1.1 Standalone Extractors

Extractor (1): Roy et al. (2015) describe a framework for quantity reasoning, consisting of two phases: extraction of Quantity-Value triples (value, units, change), and reasoning in the form of quantity entailment. In *Quantity-Value Representation (QVR)*, which is inspired by Forbus (1993) and his qualitative process theory, a quantity is defined as a triplet (v, u, ch), where v is a numeric *value*, u is a noun phrase or *unit* that describes the value, and ch specifies how the value is *changing*. The

[1] For a more extensive discussion, we refer the reader to the evaluation of quantity extractors in Chapter 4.

definition of quantity by Roy et al. (2015) is the only one among the standalone extractors that go beyond unit and value. In Chapter 4, we extend this definition further by incorporating the contextual information in the form of a *concept*.

Quantity extraction is performed in two stages: segmentation and standardization. Segmentation refers to finding segments of contiguous text that describe quantities. The authors use a *Semi-CRF (Conditional Random Fields)* (Sarawagi and Cohen, 2004) and a bank of classifiers (Punyakanok and Roth, 2000) with a set of defined word level and character level features for this task. Standardization modules are rule-based systems that convert written numbers, ranges, and dates to standardized decimal formats. In their study, Roy et al. (2015) point out the difficulty of quantity detection and ultimately use a *semantic role labeler* [2] to find the segments related to the quantity in text. Their quantity extractor is released as part of the CogCompNLP [3] package in Java, which collects several core libraries for NLP developed by the Cognitive Computation Group.

The focus of the paper, however, is not on quantity extraction but quantity entailment as a 3-way decision problem between *entails* (there exists a quantity in the given passage that entails the hypothesis quantity), *contradicts* (the quantity in the passage contradicts the hypothesis quantity) and *no relation* (there are no comparable quantities). To discover entailment relationships, the authors apply a reasoning step after quantity extraction, in which they compare quantities to see whether hypothesis quantity can be derived via some truth-preserving transformation of quantity in a passage. For a quantity to contradict or entail another, their units must be comparable, and the authors propose some rules for comparing scientific and non-scientific units.

Extractor (2): There exist two prominent open-source packages that perform quantity extraction. One tool is *quantulum3* [4]. Quantulum3 employs a combination of rules and regular expressions to extract and normalize quantities, maintaining an extensive list of scientific units and their normalized forms, along with various surface forms found in the text. To resolve ambiguities between units with similar surface forms, Quantulum3 utilizes a classifier based on GloVe vectors (Pennington et al., 2014) to determine the correct unit. The classifier is trained on the text of Wikipedia articles describing the different scientific units to learn the context in which they appear. An example of an ambiguous unit is "pound", which can refer to either a currency or a weight measurement. Such examples can only be distinguished using context information. We also adapt a similar approach for unit disambiguation in Chapter 4.

Extractor (3): Another tool worth noting is *Microsoft Recognizers-Text*, [5] which is an open-source package by Microsoft for recognition and resolution of numerical and temporal entities (Maiya et al.,

[2] A tool that assigns specific semantic roles, such as value, unit, and change, to words or phrases in a sentence.
[3] Libraries for Natural Language Processing developed by Cognitive Computation Group: https://github.com/qiangning/illinois-cogcomp-nlp (last accessed: 02.05.2024)
[4] Quantulum3 package: https://github.com/nielstron/quantulum3 (last accessed: 02.05.2024)
[5] Recognizers-Text package:https://github.com/microsoft/Recognizers-Text (last accessed: 02.05.2024)

2015; Huang et al., 2017). They provide support for more than 10 languages but are more focused on the English language. The package is written primarily in .NET but has a wrapper for other programming languages such as Python and Java. Recognition is mainly rule-based and is performed with the help of set regular expressions, which are defined separately for the five quantity types supported. Specifically, the focus is only on percentages, age, currencies, dimensions, and temperatures. Another shortcoming of this tool is that for each quantity type, a separate model is used and the quantity type has to be defined before extraction.

Extractor (4): Foppiano et al. (2019) present *Grobid-quantities* for extracting and normalizing measurements from scientific and patent literature. Their method focuses solely on the extraction of quantities with scientific units. Foppiano et al. (2019) introduce quantities as measurements that link an object or a substance with one or more quantities. Measurements take four types: *atomic*, for a single measurement (e.g., 10 grams); *interval* (e.g., from 3 to 5 km); *range* (e.g., 100 ± 4 mm) for continuous values, and a list of discrete values. A quantity consists of a quantitative value and a unit. Quantity extraction is performed in three steps: tokenization, extraction, parsing, and normalization. In tokenization, text is split by punctuation marks, then each resulting token is re-tokenized to separate digits and alphanumeric characters. Extraction is performed by labeling the tokenized data with three *Conditional Random Field (CRF)* models in a cascade. The *Quantities CRF* model determines appropriate unit and value tags and the processed results are fed into *Units CRF* and *Values CRF* to highlight known units or prefixes and to unify the format of identified values. Finally, parsing and normalization are performed to normalize and map the extracted unit surface form to one of the base SI units (Newell and Tiesinga, 2019).

3.1.2 Dependent Extractors

In this section, we describe the extractor components, which are part of a larger system, focusing specifically on the quantity extraction component. If the downstream application is relevant to the topic of this thesis, it will be detailed later. However, most of these systems are very limited in terms of quantity extraction capabilities, are tailored towards the specific downstream application, do not publish their code and often perform no evaluation on the quantity extraction component. Nonetheless, the involvement of many different researchers across various domains, each developing their own methods for quantity extraction, highlights the importance of this task.

Extractor (5): Banerjee et al. (2009) address quantity extraction as part of their ranking system using the JAPE engine (Cunningham et al., 2013) for entity search. Their objective is to aggregate information from multiple snippets to answer questions about an entity in the database with a tight quantity interval distilled from evidence of relevance in thousands of snippets. We discuss their re-

trieval method in Section 3.2.1. As for their quantity extraction, a quantity scanner annotates character spans that are likely to be quantity mentions. The scanner is a set of 150 hand-crafted rules, which are specific to the set of units and quantity types used in their study, covering mass, mileage, power, speed, density, volume, area, money, time duration, time epoch, and temperature.

Extractor (6): In the MQSEARCH system (Maiya et al., 2015), quantity extraction is performed in order to populate a numerical index. A rule-based extractor is utilized on individual sentences that populate the quantity representation in the form of *(sign, number, error, scientific notation, units)*. For example, the quantity for the measurement of gravity's curvature as "$(1.3999 \pm 0.003) \times 10^{-5} s^{-2} m^{-1}$" is extracted as "$(< empty >, 1.3999, 0.003, 10^{-5}, s^{-2}m^{-1})$." The set of rules for extraction falls into the four categories of pre-processing, units, quantities, and post-processing. Pre-processing rules identify variations in characters like minus signs, multiplication signs, and other symbols in the text and perform the necessary normalization. Unit rules are defined to identify optional prefixes for submultiples and multiples, e.g., "μ" before "meter' or "kilo" before "meter". The authors create an ontology of units from the OBO Foundry [6] and other public sources, which are used to detect units in the text. Other sets of rules are designed to detect numerical values with both an error range and scientific notation, e.g., "$30,000$" is a simple form with no scientific notation or error range, and "$1.4 \pm 0.003 \times 10^6$" contains both an error range and scientific notation. There are also rules for detecting numbers with exponents, e.g., "1.23×10^5". Finally, post-processing rules reject false detection of numerical values as quantities. We discuss the retrieval aspect of this work in Section 3.2.1.

Extractor (7): Alonso and Sellam (2018) look at quantitative information in social networks, specifically tweets. They propose a method to extract quantitative information in order to perform several experiments and analyses with Twitter data. They introduce a notion of *quantfrag* as the unit of information, which is a small piece of text that mentions one or more quantitative properties of an event. Basically, a *quantfrag* is a phrase in a text that contains just the necessary information to understand the quantity and the property being measured. These fragments are detected using rules based on language grammar and part-of-speech tags. More specifically, the focus of this work is less on the extraction and normalization of values and units but more on identifying text snippets with valuable quantitative information.

Extractor (8): Ning et al. (2022) focus on event-centric spatiotemporal quantity extraction on three specific domains of the COVID-19 pandemic, Black Lives Matter protests, and 2020 California wildfires. They are the first to tackle the quantity *detection* task using deep neural networks, by framing it as a span detection problem and utilizing the BERT model (Devlin et al., 2019a). The authors have a more restricted definition of quantities, which are special types of numbers that are

[6] Open Biological and Biomedical Ontology Foundry: https://obofoundry.org/ (last accessed: 02.05.2024)

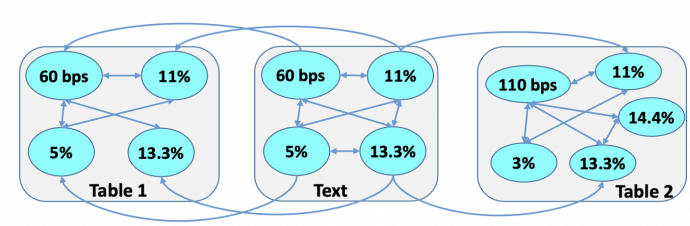


Figure 3.1: Graph of quantities in table and text for disambiguation; taken from Ibrahim et al. (2019).

associated with events, either as digits (e.g., "123") or in words (e.g., "one hundred twenty three"). Since the focus in on detection of the quantity spans, normalization of units and standardization of values are disregarded. Any temporal information such as date and time and implicit quantity information, such as ordinal numbers (e.g., "the fifth case in Seattle") and article words (e.g., "a" and "an"), are removed from the study. In addition to quantity detection, the system performs typing (detect the type of a real-world event among the three mentioned domains), spatial grounding (ground real-world events to a locale), and temporal grounding (ground each real-world event to a time). Quantities are a small part of this study, and the main focus is on events.

Extractor (9): In addition to unstructured text, quantities appear in structured formats like tables. The BriQ algorithm (Ibrahim et al., 2019) aligns quantity mentions in texts with the ones in tables. Quantity mentions in a text often contain values from table cells, but also include phrases that refer to the aggregation of multiple values. This task is harder than entity linking, since the format of quantities in text and tables might differ drastically, e.g., the text states "37K EUR" while the table refers to the same value only as "37", since the unit and the postfixes of magnitude, e.g., "K", are part of the columns. Moreover, quantities in text are often the result of summation, subtraction, and maximum or minimum of different cells. Parsing tables and their format are also another challenge in this domain. In this thesis, we only focus on quantities inside the unstructured text and do not consider tables, however, quantities play an important role in table understanding.
BriQ proposes a hybrid algorithm, with a supervised classifier that accepts or drops mention-cell candidate pairs and an unsupervised, random-walk-based model for final inference. For classification, they use a random forest classifier with a variety of textual features and quantity features based on proximity and scale. For resolving ambiguous cases, a quantity graph for values in tables and text is created, and a random walk-based method is used for resolution. Figure 3.1 shows an example of such a graph, where in-text mentions of "11%" and "13.3%" have exact matches in two different tables. The stationary visiting probabilities of the random walk on the graph are used as scores for disambiguation and correct alignment. Their work is used to create ExQuisiTe (Ibrahim and Weikum, 2019), a system to align quantity mentions in text with related quantity mentions in tables.

Table 3.2: List of quantity-aware retrieval systems, their query formulations, output representation, and summary of the method.

Model	Query	Output	Method	Data Type
1. Ho et al. (2019)	(quantity type, context, condition)	Qfact of (entity, quantity, context)	bi-LSTM for extracting Qfacts, filtering on conditions, ranking context with probabilistic/embedding methods	general
2. Li et al. (2021)	(quantity, context, entity type, condition, time condition)	table or charts	results are filtered by entity type, quantity, and time condition, then aggregated in charts	general
3. Banerjee et al. (2009)	(context, unit)	quantity intervals	intervals are tagged in text and ranked with frequency/lexical features and RankSVM	consensus data
4. Sarawagi and Chakrabarti (2014)	(context, unit)	quantity intervals	intervals are extracted from tables, their distributions are computed with density estimators	consensus tables
5. Ho et al. (2021a)	(quantity type, context, condition)	Qfact of (entity, quantity, context)	extracting Qfact with row and column alignments, filtering based on conditions, ranking with embeddings	tables
6. John K. Stockton (2009)	(context, range, quantity)	documents	unit-based index structure	general
7. Maiya et al. (2015)	(property, range, quantity)	documents	values and units in an index	general

Extractor (10): Sarawagi and Chakrabarti (2014) describe a system that performs quantity extraction on tables for question answering. In this case, a probabilistic context-free grammar is used, which exploits co-occurrence statistics between quantity types, units, and phrases from an unlabeled corpus of tables. Given that the quantity representation in tables is different from the one in natural text, they take advantage of column and row headers for unit normalization and design a specific unit annotator for table columns. From the values in tables, a distribution for distinct quantitative attributes of entities is created. These distributions are then used to answer queries. The goal is to answer queries regarding attributes of an entity, e.g., "average revenue of Microsoft" and "half-life of plutonium", with a distribution derived from the tables in the corpus.

3.2 Quantity-Aware Retrieval Models

Related work that directly addresses the problem of quantity in search and retrieval is limited. There is no proper definition of a quantity-aware retrieval system, and models that do both retrieval and deal with quantities have various output representations. Some focus on factual information that contains quantities (Ho et al., 2019, 2020, 2021a), while others focus on the aggregation of quantities in tables or text based on common property and report distribution of values (Banerjee et al., 2009; Li et al., 2021). Retrieving text snippets that impose certain quantity constraints are more studied in context of index structures and patent documents, where a numerical *filtering* option is added to a term-based index (Agrawal and Srikant, 2003; Fontoura et al., 2007; Maiya et al., 2015). However, the methods proposed for handling quantities in index structures primarily focus on storage, efficient document access, and filtering out irrelevant documents, rather than ranking.

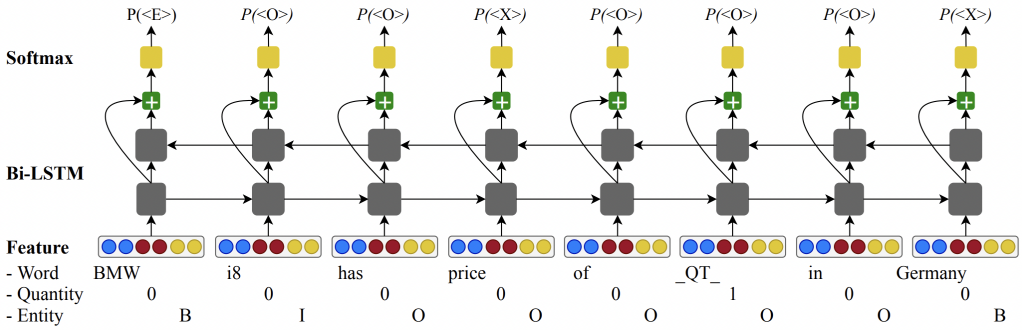


Figure 3.2: Qfact extraction with semantic role labeling; taken from Ho et al. (2019). The position of the entity and quantity are encoded as input with word embedding to predict Qfacts in a text.

Table 3.2 gives a summary of the different methods with the input and output representations. The methods are numbered for easy reference to their descriptions in the following text. The output representation varies across different methods, making the methods practically incomparable to one another. Moreover, there is no available benchmark dataset for this task. Index-based methods return paragraphs or documents, whereas retrieval systems either aggregate the quantities in intervals and charts or abstract the quantity information in *Quantity Facts (Qfacts)*. Qfacts are facts about an entity that involves quantitative information, e.g., " heights of towers", "running times of athletes", or "energy consumption of a car".

Models that output an interval aim to find a distribution of values related to a property of an entity, e.g., "body mass". In contrast, systems that output Qfacts return snippets that support specific quantity conditions (e.g., greater than, less than, within a range, or equal to a certain value). In this thesis, our approach to quantity-aware retrieval aligns more closely with the latter variant, where we return text snippets that satisfy specific numerical conditions rather than extracting Qfacts.

One common element among all systems is that they rely on the correct extraction of values and units, in other words, their performance is dependent on the quantity extraction phase, highlighting once again the importance of quantity extraction.

Another point worth noting is the lack of neural retrieval models in this domain. Word embeddings that are used for ranking are mainly non-contextualized, like Word2Vec, and matching is performed mainly on lexical features. Ho et al. (2019) use a bi-LSTM model for semantic role labeling but for the final ranking a static embedding and bag-of-word model is utilized. Although there are quantity-specific language models and word embedding none of them are used for retrieval purposes. We discuss a few of these systems in Section 3.3.

3.2.1 Retrieval Models

Retrieval Method (1): Ho et al. (2019, 2020) were the first to focus on the retrieval of entity-centric Qfacts. They introduced *Qsearch*, a system designed to model Qfacts associated with named enti-

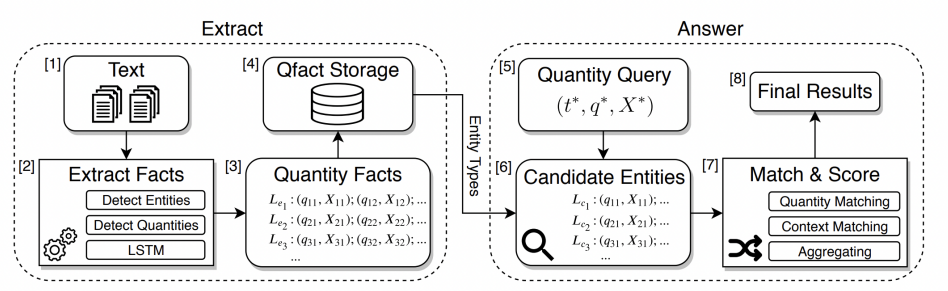


Figure 3.3: Overview of Qsearch from Ho et al. (2019).

ties. Each Qfact is represented as a triple *(entity, quantity, context)*, where the quantity includes a numerical value, a unit, and a value resolution (exact, approximate, upper/lower bound, interval). The context is a bag-of-words describing the relation between the named entity and the quantity. A bi-directional LSTM (Hochreiter and Schmidhuber, 1997) detects the relevant spans for Qfacts from the text through semantic role labeling. Each token in the text is annotated with the roles Entity, Context, and Outside token.

Prior to semantic role labeling, quantity extraction is performed using the quantity extractor from (Roy et al., 2015) and named entities are detected and disambiguated using AIDA (Hoffart et al., 2011), which links named entities to the YAGO knowledge base (Suchanek et al., 2007). Figure 3.2 shows the semantic role-labeler, with word embeddings as word features and annotations for quantity and named entity positions for quantity and entity-level features. The semantic role labeler is trained using a semi-supervised approach. The training data for Qfacts are generated in three stages and used as ground truth. First, quantities are identified and extracted, followed by the detection and disambiguation of named entities. Finally, Qfacts are generated with relation extraction (Angeli et al., 2015) between quantities and named entities in a sentence. These generated Qfacts are used to train the semantic role labeler, which replaces the relation extraction module.
After training, Qfacts from all the documents in the corpus are extracted and indexed for ranking. During ranking, the query is processed to identify the entity, quantity, context, and numerical condition. An example query for such a system is “Hybrid cars with a price under 35,000 Euros”, where the entity is “Hybrid cars”, the quantity is “35,000 Euros”, the numerical condition is “less than” and the context is any remaining token. The processed query is scored against the extracted facts that match the entity and quantity types from the query. In the mentioned example, text snippets that contain named entities with the type “car” and contain currencies as quantities are selected as candidates. Either a *probabilistic* or *embedding-based* ranking of the query context against candidate Qfact contexts determines the final ranking The ranking mechanisms primarily differ in their approach to context matching: the probabilistic method employs a traditional approach by comparing bag-of-words representations, while the embedding-based ranking performs soft matching between words with similar meanings.

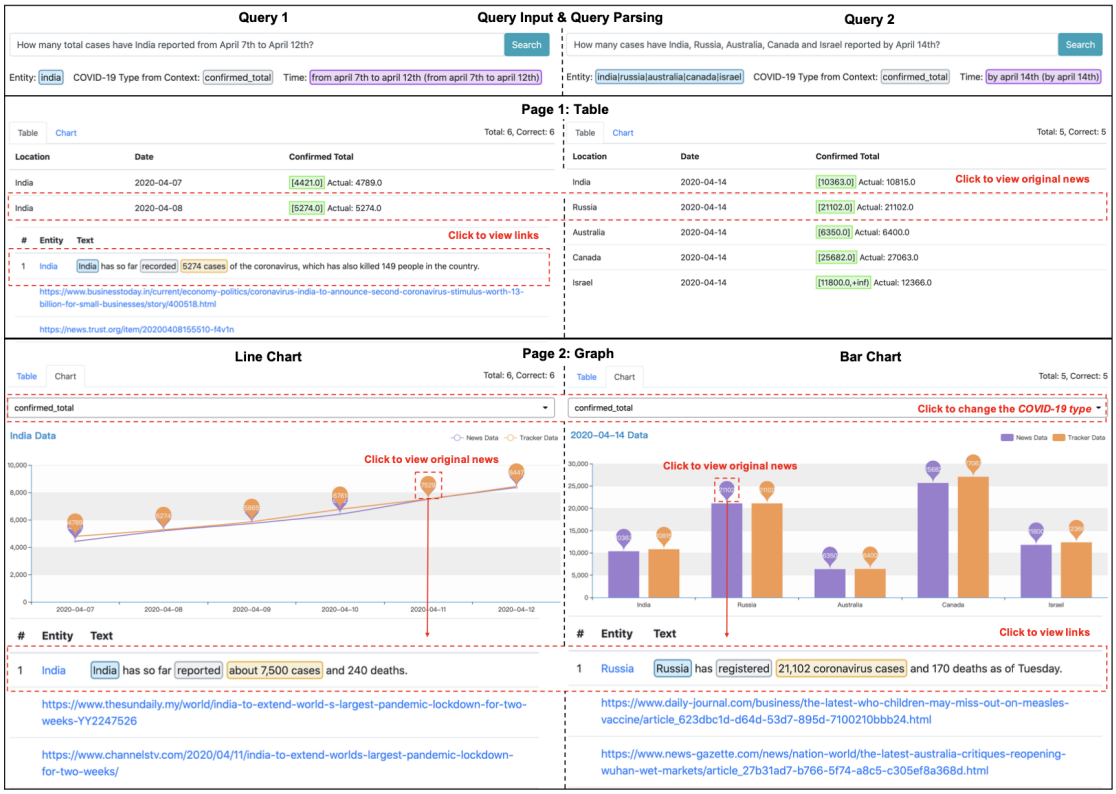


Figure 3.4: Visualization of the results from AnaSearch for example queries (Li et al., 2021).

Figure 3.3 gives an overview of the Qsearch architecture, where a corpus of documents is processed to extract Qfacts and to store them in a database. The stored Qfacts are used in the *Answer* phase, to answer queries with a numerical condition and specific entity type. The authors further provide a demo (Ho et al., 2020) for interacting with their system but refuse to provide the source code or their training data, making their approach difficult to evaluate.

One downside of Qsearch is that components of the pipeline, such as semantic role labeling and named entity extraction, are expensive and rely on scarcely available annotated data for quantities. Moreover, this method is tailored towards named entities, which many quantity-centric queries may not contain. Nonetheless, this is the only model that is comparable to our work. Unfortunately, the authors neither provide their code, nor their training data. Multiple tedious efforts were made to implement their system from scratch, however, not only does the data generation system depend on outdated and deprecated libraries but also the training and evaluation proved to be extremely inefficient in practice. In this thesis, we propose a computationally cheaper option by eliminating the need for explicit entity detection or semantic role labeling or expensive data annotation.

Retrieval Method (2): AnaSearch (Li et al., 2021) is a demonstration paper for analytical query processing on unstructured text. The output of analytical queries is structured into tables or visualized with charts. AnaSearch consists of three parts: quantitative data extraction, query parsing and re-

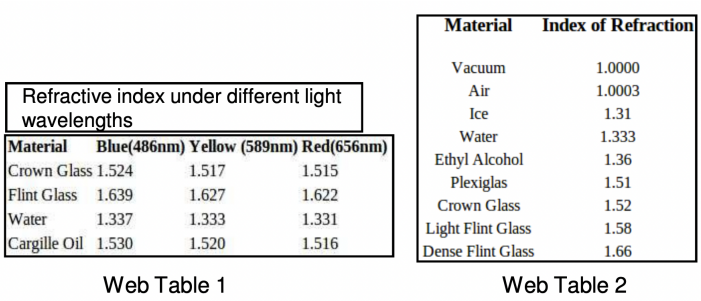


Figure 3.5: Sample tables responding to query "refractive index of flint glass"; image taken from Sarawagi and Chakrabarti (2014).

trieval, and data visualization. An example of the output as a bar chart and line chart is shown in Figure 3.4. Similar to Qsearch, AnaSearch performs Qfact extraction, where a given query is ranked against the facts extracted from the corpus. The extraction method utilizes a constituency parsing tree to produce triplets (*related, value and unit, time*). The *related* component includes information about the quantity being measured and any entities influencing the value. The *value* is the number in the text, along with its value resolution and corresponding unit. The *time* denotes when the quantity assumes a specific value, enabling visualization of value changes over time.

Based on the extracted facts, AnaSearch supports two types of queries: entity-based queries and quantity-based queries. An entity-based query primarily focuses on retrieving facts that match the entities specified in the query. A quantity-based query, similar to those in the Qsearch system, includes an additional time dimension. For example "Sales of Tesla Model S and BMW i8 in 2019" is an entity-based query, with "Tesla Model S, BMW i8" as the entity condition and "2019" as the time condition. "Sales" serves as the context. In contrast, the query "cars with sales over 5,000 in April", is a quantity-based query as it includes a quantity condition of "more than 5000". In this case, "cars" is the condition on entity type, and the month of "April" is a time constraint. Queries in Figure 3.4 are both entity queries since none of them contains a numerical condition.

While matching quantities against a condition, the retrieved unit should be comparable and the value should satisfy the quantity condition. Due to the temporal dimension, the retrieved facts should also satisfy the temporal conditions. The matching Qfacts are scored based on the occurrences in the corpus and grouped based on entities or time to be visualized in charts or tables to enhance the analysis experience.

Similar to Qsearch, AnaSearch focuses on Qfacts with a more strict definition of entity and quantity relation and does not explore generic nouns and search terms. Additionally, in their demo system, no evaluation of the performance or comparison with previous methods is mentioned. The focus is mainly on innovative visualizations.

Retrieval Method (3): Banerjee et al. (2009) consider quantities as a class of named entities and introduce *Quantity Consensus Queries*, where the answer is a quantity interval distilled from the quan-

tities gather in many relevant text snippets. Their method focuses on scoring quantity intervals and combining snippets of quantity and text information. Unlike our approach, their queries do not contain numeric information. Quantity consensus queries express uncertainty about the answer quantity, e.g., "driving time from Paris to Nice" or "battery life of Lenovo X30", and they aim to learn a reasonable distribution over the uncertain value. The distribution is estimated by aggregating evidence in favor of candidate quantities from snippets or segments of text around the mentioned entity that matches the unit specified in the query. The output of the system is a consensus interval, a tight range of quantities that has strong collective support from high-scoring snippets. The authors define a set of frequency-based and lexical-based features and rank intervals in snippets with RankSVM (Joachims, 2002) and weighted majority votes.

Retrieval Method (4): Sarawagi and Chakrabarti (2014) describe an approach for finding census queries on tables. A quantity query has two parts: The first part is a description of the quantity type, e.g., "distance", "speed" or "emission", and the second indicates an entity, e.g., "Pluto", "Microsoft", and "Apple". To determine the distribution of a desired attribute for an entity, a set of tables is retrieved that match the entity and quantity type. Figure 3.5 shows an example of tables retrieved for the query "refractive index of flint glass". A quantity extractor detects the unit and values in the table snippet, from which a distribution over the target attribute of the query entity is formulated. Although Sarawagi and Chakrabarti (2014) and Banerjee et al. (2009) rank quantity information to generate value distributions, this is not their primary objective. Additionally, their query language and output representation differ significantly from those used in retrieval systems.

Retrieval Method (5): Ho et al. (2021a) introduce *QuTE*, a method for automatically extracting Qfacts from web tables. These facts are used to generate answers for quantity-filter queries. There is also a demo of their work available (Ho et al., 2021a,b). The quantity-filter queries are similar to Qsearch, where a numerical condition is imposed on a quantity related to an entity, e.g., "Skyscrapers higher than 1000 feet".
First, quantities are extracted and normalized using pattern-based extractors and rule-based normalization. Quantities are then enriched with contextual information from tables through rules applied to the HTML DOM tree. The pipeline begins with recognizing and normalizing quantities in the table's quantity-column and linking entities in the entity-columns to a knowledge base. This column alignment produces a Qfact. Additional context from the surrounding table and statistics from external corpora are used to further enrich these facts. To find Qfacts relevant to a query, the database of all facts is first filtered based on the entity type and quantity values that match the numerical condition. Finally, an embedding distance between the context of the query and Qfacts in the database ranks the elements of the final set. Similar to Sarawagi and Chakrabarti (2014), this work focuses on web tables and not unstructured text.

3.2.2 Numerical Indicies

Before quantity ranking gained attention in the information retrieval community, database research had already explored numerical indices. These indices are made specifically to support queries that contain numerical restrictions (Agrawal and Srikant, 2003; Fontoura et al., 2007; Maiya et al., 2015) and patent documents also mention them (Burrows, 1996; John K. Stockton, 2009; Stepanova et al., 2023; Marcus Fontoura and Zien, 2004). However, the primary focus of these indices and patents is not the contextual *ranking* of quantities in text, but rather the efficient structuring and filtering of results based on strict numerical constraints. For example, with an *equal* constraint, only the rows in the database that match the exact value will be returned. If no exact match exists, the result set will be empty, as no proximity matching of nearby values is performed. This creates a hard constraint filter rather than a ranking system. Some patent documents introduce a numerical posting list (Burrows, 1996; John K. Stockton, 2009; Marcus Fontoura and Zien, 2004), where each list is associated with a range of values and includes document identifiers for those containing values within that range. The generated posting lists are multi-layered and stored to answer queries based on a range of values. Documents are returned if they fall within the posting lists containing the minimum, maximum, and in-between values specified in the query. The primary focus is purely on constraint queries involving values, without considering the context or the units of the quantities. From the body of work on numerical indices, we mention two index structures that take context information into account and are therefore more relevant to our work.

Retrieval Method (6): John K. Stockton (2009) focus on contextualizing numeric data by including both a numeric value and an accompanying unit. The method includes a *unit-based index* construction, where an indexer extracts numeric data from a set of source documents and converts number-unit pairs from natural text or tables into number-unit tokens. Additionally, standardization and unit conversion are performed. Other metadata and keywords from documents are also stored in an index with positional information, resulting in a more contextualized index structure than solely numeric values. Utilizing such an index, the user can formulate queries that contain numerical conditions and filter results accordingly.

Retrieval Method (7): The work of Maiya et al. (2015) also uses a numerical index to answer queries on measured information, in a framework called *MQSEARCH*. The framework is a facet-based navigation system for measured quantities, measured properties, and the topics and themes with which they are associated. Quantities are saved in an index structure as a 5-tuple of the form *(sign, number, error, scientific notation, units)*. The paper itself is mainly focused on the quantity extraction part, and the details of facetted navigation or how the index structure is realized are not mentioned. It appears that they rely on the Apache Solr built-in functionalities for this purpose. [7] The framework

[7] Solar, open source search platform: https://solr.apache.org/ (last accessed: 02.05.2024)

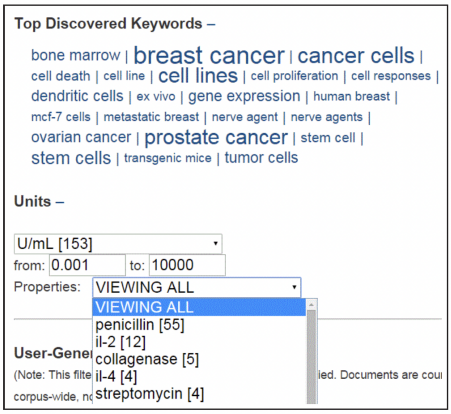


Figure 3.6: Choosing "U/mL" from the dropdown list reveals the associated topics (e.g., "breast/prostate cancer") and properties (e.g., "concentrations of penicillin"). Additionally, ranges can be defined (i.e., "0.001 to 10,000"). Figure is taken from Maiya et al. (2015)

is only evaluated on the task of quantity extraction on the random number of sentences from unclassified research reports from the U.S. Department of Defense, where the data is not published. Figure 3.6 shows an example of their query structure, where the user has to first choose a unit of measurement, to filter for associate keywords and properties in the neighboring dropdown list. In addition to the unit, a user can further define desired ranges for values.

3.3 Embeddings, Language Models, and Quantity Understanding

As discussed in Chapter 2, language models and embedding techniques are fundamental to many NLP tasks, including question answering and IR. However, quantities are underrepresented in typical textual corpora used for pre-training these models. Moreover, the main benchmark datasets for evaluating these representations do not focus on quantity understanding. As a result, quantities are often neglected and the numeracy abilities of these models are limited. In recent years, efforts have been made to create dedicated embeddings for specific tasks or token types, such as named entities in text (Almasian et al., 2019; Cherry and Guo, 2015). These efforts have also extended to the development of quantity-aware language models and embeddings.

In this section, works that identified the issues with numeracy in dense representations, through probing tasks and domain-specific datasets, are discussed. Then, a number of contextualized and non-contextualized embedding techniques that aim to circumvent these problems are introduced. Table 3.3, shows a summary of the embedding methods described in this section. The first group are the non-contextualized or static embeddings, the second group are contextualized models and the last group are domain-specific language models. The last group differs from the rest as their main focus is on training a corpus with more numerals instead of adapting the architecture. Spithourakis and Riedel (2018) were the first to design a specific language modeling objective for numbers. Later, NumGPT (Jin et al., 2021) uses the finding from the static numerical embeddings in Jiang et al.

Table 3.3: List contextualized and non-contextualized quantity-aware embeddings.

Model	Name	Method Summary	contextualized
1. Sundararaman et al. (2020)	DICE	cosine similarity between word embeddings of two numbers should reflect their distance on the number line, define a linear mapping between similarity and distance	✗
2. Jiang et al. (2020)	-	aims to solve the OOV problem with prototypes, which are typical numbers chosen by a clustering method, embedding a number is the weighted average the prototypes weighted based on similarity	✗
3. Spithourakis and Riedel (2018)	-	RNN-based language model that predicts numbers differently, either by binning them into finite vocabulary, predicting digit by digit with an RNN, or mixture of Gaussians	✓
4. Jin et al. (2021)	NumGPT	add an extra number embedding to GPT, that is the combination of mantissa and exponent in scientific notation, the mantissa is computed with prototypes	✓
5. Berg-Kirkpatrick and Spokoyny (2020)	-	introduce (1) masked number prediction for predicting a missing number and (2) numerical anomaly detection for detecting an errorful number in a sentence	✓
6. Spokoyny et al. (2022)	GeMM	mask numbers and unit during language modeling, called Masked Measurement Prediction (MMP)	✓
7. Petrak et al. (2023)	-	arithmetic-base pre-training with contrastive learning to improve number representation with the addition of inferable number prediction objective to improve numeracy sentence	✓
8. Araci (2019)	FinBERT	BERT model trained on financial data	✓
9. Loukas et al. (2022)	FiNER	either replaces numbers with the [NUM] token or sudo-token reflecting their shape	✓

(2020) to design a transformer-based language model capable of handling numbers. More recently, masking quantity tokens, numbers, and units have shown improvements in the numeracy of language models (Berg-Kirkpatrick and Spokoyny, 2020; Spokoyny et al., 2022).
Although we do not use numerical embeddings and language models to create our quantity-aware retrieval models, covering related work in this area is important. First, studying the shortcomings of these models helps us understand the limitations of current retrieval systems in handling quantities. Second, efforts in this direction offer inspiration for integrating quantity understanding into retrieval systems. To this end, we first start by looking at probing tasks and works that investigate the shortcomings of embedding methods and language models when it comes to numerical data. Then we look at a number of non-contextualized or static embedding methods for numbers and conclude with a numerical language model or contextualized embeddings.

3.3.1 Investigating Numeracy and Probing Tasks

We begin by investigating the issues with prominent word embedding techniques for numerical representations through a series of probing tasks. These tasks are designed to test whether an embedding of a number is suitable for tasks involving numerical understanding. This is achieved either by checking the representations independent of the surrounding context, i.e., whether it has a notion of magnitude, or in the context of textual content, i.e., filling in the blanks for common sense knowledge. Table 3.4 shows an overview of the main works in this domain and the probing tasks studied in the works. The probing tasks and the main findings of each work are organized in columns, among which the work of Thawani et al. (2021) is more of a survey of possible probing tasks and does not introduce a new one. Almost all papers point to the same conclusion that the popular tokenization and training strategies for words do not extend well to numbers. There are also certain rules of thumb for better numerical representations that are useful to create quantity-aware models. We

Table 3.4: List of works investigating numeracy in embeddings and their main takeaway messages.

Citation	Tasks	Findings
1. Chen et al. (2019a)	(1) predict a range of magnitude in text (2) detect exaggerated numbers in text	(1) RNN models better than a CNN for contextual embeddings (1) bi-directionality enhances performance (3) minor distortions in values are hard to detect
2. Naik et al. (2019)	contrastive test for magnitude and numeration: (1) One-vs-all (2) Strict contrastive (3) Broad contrastive	(1) approximate notion of magnitude is present but not a precise one (2) numeration is not captured in current embeddings
3. Wallace et al. (2019)	set of probing tasks for numeracy: (1) list maximum (2) decoding (3) addition	(1) character-based models work better (2) embeddings work well for ranges in training but fail to extrapolate (3) sub-word tokenization not suitable for numbers (4) capabilities are stable across various number types (5) right padding keeps the digits in the correct positions
4. Zhang et al. (2020)	given an object and attribute predict the distribution of values with either (1) regression, or (2) multi-class classification into buckets	(1) multi-class classification beats regression (2) numerical representation has an impact on scale prediction (3) scientific representation with mantissa and exponent is superior (4) contextualized models are better than non-contextualized ones
5. Lin et al. (2020a)	commonsense probing through language modeling	(1) language models do not preserve relationships between objects and numbers during pre-training (1) small perturbation in context changes the output
6. Thawani et al. (2021)	(1) simple arithmetic (2) numeration (3) magnitude comparison (4) arithmetic world problems (5) exact fact (6) measurement estimation	(1) scientific notation is the most promising (2) log scale works better than linear scale (3) character-level tokenization is better

summarized these findings in Table 3.4.

Probing Task (1): Numeracy-600K (Chen et al., 2019a) was the first benchmark dataset provided for testing models in their ability to predict the magnitude of a number at some specific position in text. For example in the sentence "S&P 500 <.SPX> up 1.53 points at ___ after market open". A financial investor would know from the context that the blank should be filled with quotes of the index "S&P 500", with a 4th-magnitude numeral, e.g., "1840". The model with numeracy capabilities should detect the target entity in a sentence, and understand the type of information to insert into the blanks. The dataset is designed in a way that each numeral is separated into eight classes by magnitude and a proposed model should predict a suitable range. The data is collected from 600K market comments from Reuters, [8] hence the name of the dataset. In addition to the data, result of testing seven different architectures for contextualized word embeddings are reported in the study, including CNN (Kim, 2014), GRU (Cho et al., 2014a), BiGRU, CRNN (Choi et al., 2017), CNN-capsule (Sabour et al., 2017), GRU-capsule, and BiGRU-capsule (Wang et al., 2018). BiGRU and GRU-capsule models were the best-performing ones. The best model's reasoning ability is further tested by multiplying the numerals in the text by different distortion factors, where the model should detect whether a numeral is correct, overstated or understated. One finding of the paper is that if a numeral is distorted by a large factor most models could easily recognize it, while minor distortions were harder to detect. This dataset is used in most works mentioned later in this section to evaluate

[8] News website: https://www.reuters.com/ (last accessed 02.05.2024)

for magnitude understanding of embeddings and language models.

Probing Tasks (2): The work of Naik et al. (2019) looks at the popular models at that time (2019), namely, Skip-gram, GloVe, and FastText, where numbers make up less than 5% of the vocabulary. An analysis framework is developed for testing if these non-contextualized embeddings capture magnitude (e.g., $3 < 4$) and numeration (e.g., $3 = three$). Since numbers follow a well-defined ordering, independent of their textual context, this ordering should be preserved in the embedding space. To this end, three categories of contrastive tests are proposed. A contrastive test for a property p is defined as a triple (x, x_+, x_-) such that x is closer to x_+ than x_- under p. Here, p is the tested property of either *magnitude* or *numeration*, resulting in the following tests:

- *One-vs-All*: $x_- = \{y | y \in X - x, y \neq x_+\}$ the model should find x closer to x_+ than to all x_-, where X is the list of numbers in the vocabulary. An example for magnitude is $(3, 4, x)$, such that $x = \{y | y \in X - \{3\}, y \neq 4\}$ and the embedding of 4 should be closer to the embedding of 3 than all other integers. For numeration, an example is $(3, three, x)$, such that $x = \{y | y \in Y, y \neq three\}$, where the embedding of $three$ should have a minimum distance in comparison other embeddings in the vocabulary.

- *Strict Contrastive*: Choose x$_-$ to be the second-closest to x after x_+ under p. An example for magnitude is $(3, 4, 5)$, where 4 is closer to 3 than 5, and for numeration is $(3, three, four)$.

- *Broad Contrastive*: x_- is the furthest from x under property p. An example for magnitude is $(3, 4, 1000000)$, where 1000000 is the furthest integer from 3, and 4 is closest. For numeration, an example is $(3, three, billion)$, which is the same but with spelled-out numbers.

For all experiments, cosine similarity is used as the distance metric. By observing the results of the three sets of tests under the magnitude property, the authors conclude that the models tend to work well on *Broad Contrastive* task but perform poorly on *One-vs-All* and *Strict Contrastive*. This indicates that all models capture an approximate notion of magnitude but not a precise one. A reason for this might be the variation in context, where numbers with similar magnitude usually occur in similar contexts. While the context information allows the embeddings to distinguish large-magnitude shifts, it does not provide an actual understanding of scale. The results for the numeration tasks are more disappointing, where all models except GloVe fail to significantly beat the random baseline.

Probing Task (3): Later that year, Wallace et al. (2019) designed another set of probing tasks. Various embeddings, from traditional word embedding methods, such as, GloVe (Pennington et al., 2014) to language model-based techniques, like ELMo (Peters et al., 2018) and BERT (Devlin et al., 2019a) were studied in this work. Additionally, they also introduce a character-level CNN and a character-level LSTM, as character-level tokenization is better suited for numerical representation in comparison to word-level tokenization or byte pair encodings. The authors get their inspiration

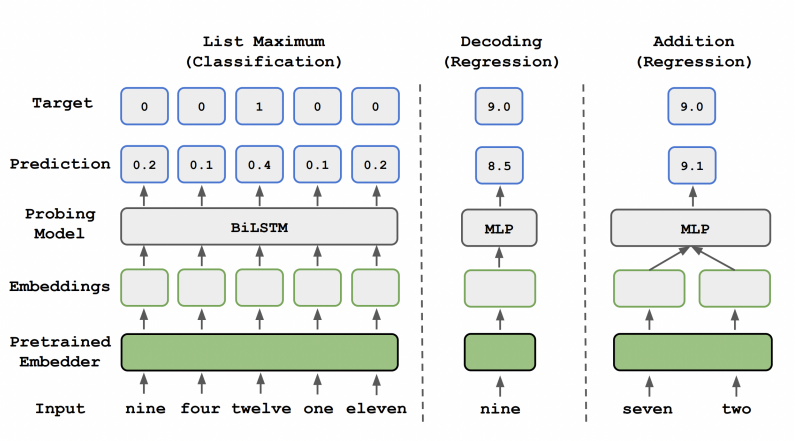


Figure 3.7: Probing setup by Wallace et al. (2019). A pre-trained embedder is used for the probing models. Probing tasks include finding a list's maximum, decoding a number, or adding two numbers. Numerical values are represented as digits ("9"), floats ("9.1"), or negatives ("-9").

from the NAQANet model (Dua et al., 2019), which is described in Section 3.4. To answer numerical reasoning questions, the NAQANet model implicitly learns to perform binary comparison (e.g., "Which country is a bigger exporter, Brazil or Uruguay?"), greater than comparison (e.g., "Which player had a touchdown longer than 20 yards?"), identification of a list maximum (e.g., "How many yards was the shortest field goal?"), and argmax relations (e.g., "Who kicked the longest field goal?"). However, the authors suspect that the source of numerical information lies in the token embeddings, i.e., the character-level convolutions and GloVe embeddings of the NAQANet model. Therefore, the probing task is focused on the token embeddings.

In the *list maximum* task, given a list of embeddings for five numbers with values of similar magnitude, the model should correctly predict the index of the largest quantity. A span selection model, an LSTM trained with negative log-likelihood, is trained for this task.

The *decoding* task probes for magnitude by regressing to a value given an embedding, e.g., given the string "five" the model should regress to 5.0. Embedding of the word "five" is passed through a multi-layer perceptron, which trained using a mean squared error loss to regress to the value 5.0.

The *addition* probing task finds the sum of two numbers given the embedding of their values, using a three-layer fully connected network, trained using MSE loss.

Figure 3.7 illustrates the architecture and an example for the three probing tasks, where the decoding and addition regress to a certain value, while list maximum is more of a classification task.

Ultimately, probing tasks show that CNNs are a particularly good prior, resulting in ELMo's superior numeracy compared to BERT. Additionally, character-based models perform slightly better

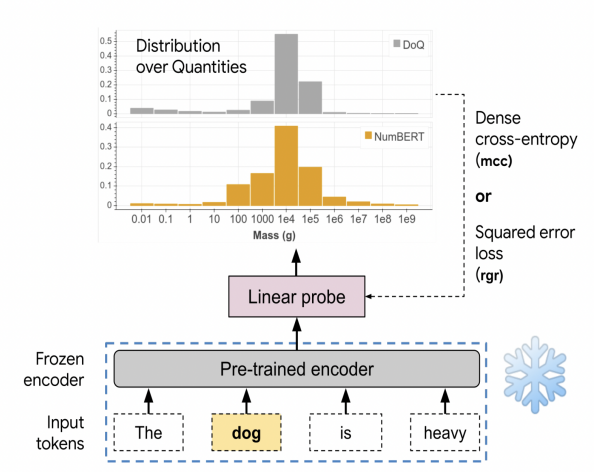


Figure 3.8: Scalar probing from Zhang et al. (2020). The mass of "dog" is a distribution (gray histogram) concentrated around 10-100kg. A linear model over a frozen encoder predicts the distribution in orange using either cross-entropy or a regression loss; the image is taken from the respective paper.

than word-level or word-piece tokenizers. However, most embeddings perform reasonably well on all probing tasks when the values fall within the ranges observed in the training set, but they struggle to extrapolate to new values.

Probing Task (4): Zhang et al. (2020) investigate if the language models capture scalar magnitudes of objects, through *scalar probing*. This task is close to common sense reasoning, such that given an object, e.g.,"car", and an attribute with continuous numeric values, e.g., "price", the model should predict possible values. Since there may not be a single correct value for each attribute, predictions are a distribution of possible values. For this task, the dataset Distributions over Quantities (Elazar et al., 2019) is used, consisting of empirical counts of scalar attribute values for more than $350K$ nouns, adjectives, and verbs, collected from web data. To predict the scalar value for nouns, a probing layer is added on top of the embedding to either regress to a point in the distribution or predict a bucket into which a value falls. An example of scalar probing is shown in Figure 3.8.
The results demonstrate that contextual word embeddings, such as BERT and ELMo, outperform non-contextual ones like Word2Vec and GloVe. Additionally, representing numbers in scientific notation during pre-training can enhance overall performance. The scientific notation is added to a contextual model like BERT by a combination of an exponent and mantissa, for example, 314.1 is represented as 3141`[EXP]`2, where `[EXP]` is a new token added to the vocabulary. With this minor adjustment, models more easily associate objects in a sentence directly with the magnitude expressed in the exponent, ignoring the relatively insignificant mantissa.
We also experimented with adding scientific notation to the quantity-aware retrieval systems described in Chapter 5. However, this approach proved detrimental in our experiments, significantly reducing performance. It is important to note that we did not incorporate scientific notation during pre-training. Our models rely on pre-trained checkpoints for general-purpose retrieval, and we

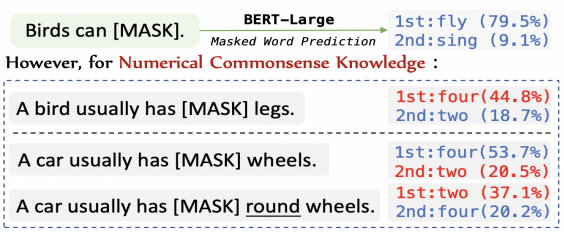


Figure 3.9: Figure from Lin et al. (2020a) show that language models fail to stay consistent under small perturbations. With an additional token the distribution shifts towards the wrong prediction.

introduced this notation only during fine-tuning. This limited exposure may have been insufficient for the model to learn the correct representation.

Probing Task (5): Lin et al. (2020a) introduce a numerical common sense reasoning probing task as well as an associated dataset. In the probing setup, a pre-trained language model aims to predict a mask quantity value from the context based on common sense knowledge in its representation. To create the dataset, the authors focus on only 12 number words, from zero to ten, and annotate training sentences based on them. Some examples from the dataset include "The world contains *seven* continents" and "Water will freeze at *zero* degrees centigrade", where common sense knowledge about the world helps with the prediction of correct values. Interestingly, powerful transformer-based language models perform poorly on this task, even after fine-tuning with distant supervision. Moreover, the language models are brittle under a simple perturbation of inserting an adjective near the masked number, e.g., changing "A car usually has [MASK] wheels." to "A car usually has [MASK] round wheels." reduces the probability of the value being predicted. An example of such behavior is given in Figure 3.9. The authors added adversarial examples by adding adjectives before the noun involved in the numerical reasoning to accommodate these perturbations.

Another surprising observation was made by looking at the attention weight of the language models when predicting numbers, e.g., by plotting the attention distribution of the sentence "A bird usually has two legs" with respect to the word "two". In this case, words that humans consider important for predicting the numerical value, namely "birds" and "legs" have lower weights in comparison to unrelated tokens. This suggests that these models do not preserve the relationship between subject/object and number words during pre-training.

Probing Task (6): Thawani et al. (2021) analyzed a myriad of representational choices made by 18 publications and are the first to define a taxonomy for numeracy tasks. Drawing from cognitive science, the authors organize numeracy tasks into two categories.

1. *Granularity:* indicating if the encoding is exact, e.g., "cows have four legs", or approximate, when a vague guess of the value is enough, e.g., "she is about 180 cm tall".

2. *Units:* indicating if the number is abstract and the raw numerical value is enough, e.g., "$2 + 3 = 5$" , or grounded by a specific unit, e.g., "5 apples".

For tasks grounded by units, both numbers and words should be understood in the same semantic space. Based on these two dimensions, the eight probing tasks are categorized as follows:

1. *Simple Arithmetic:* Tasks that involve mathematics, such as addition and subtraction over numbers, which are often solved with masking and causal language models.

2. *Numeration:* Refers to mapping a string to its numeric value, e.g., string "10" to value "10.0". This is usually done by regressing from an embedding representation to a numeric value.

3. *Magnitude Comparison:* The ability to tell which number is larger, referring to argmax of a list of numbers or binary classification between two values.

4. *Arithmetic Word Problems:* The grounded version of arithmetics that is present in school books, e.g., "Mary had two cookies. She gave one away. How many does she have left?"

5. *Exact Facts:* Adding common sense knowledge into embeddings, such as "cats have 4 legs".

6. *Measurement Estimation:* The task of approximating measures of objects along certain dimensions, e.g., "the number of seeds in a watermelon" or the "weight of a telephone".

7. *Numerical Language Modeling:* More of a setup than a task, numerical language modeling is analogous to masked/causal language modeling for words. Some tasks can be framed as numeric language modeling, e.g., "arithmetic (5+3=[MASK])" and "measurement estimation (cats weigh [MASK] kg)". However, they require different evaluation setups and regression-based metrics such as mean absolute error, root means squared error, or their log-scaled and percentage variants.

8. *Downstream Applications:* Numeracy is used in various downstream tasks from detecting sarcasm in tweets based on quantities (Dubey et al., 2019) to identifying claims in financial documents (Chen et al., 2020a), where improvements in the downstream application indicate the improvement of the underlying representation.

In Table 3.5, the mentioned eight tasks are arranged into the two categories of *Unit* and *Granularity*.

After inspecting a variety of methods and probing tasks, a set of rules of thumb and recommendations are proposed:

- If the output prediction is a string then scientific notation is more promising than decimal notation and character-level tokenization outperforms subword-level tokenization.

Table 3.5: Tasks by Thawani et al. (2021) organized into four categories.

	Abstract	Grounded
Exact	simple arithmetic: 2+3=5	arithmetic world problems: 2 hats + 2 hats = 4 hats exact facts: lions have four legs
Approximate	numeration: 2: '2' magnitude comparison: 6>4	measurement estimation: lions weigh 200 lbs

- For real value representations, log scale is preferred over linear scale, and binning (dense cross-entropy loss) works better than continuous value prediction (MAE loss).
- Design choices define the trade-offs between inductive biases and data-driven variance, by targeting a broad range or narrow range of numbers to be represented, e.g., multi-class classification over a fixed number of bins in contrast to value embeddings with continuous and unrestricted representations.

3.3.2 Non-Contextualized Representations

We start with embedding methods that non-contextualized. Contextualized embeddings are often learned in combination with language modeling tasks, to encode the surrounding context, whereas the non-contextualized ones focus on learning embeddings directly. The first two rows of Table 3.3 summarize the two methods discussed in this section.

Embedding (1): Deterministic, Independent of Corpus Embeddings (DICE) (Sundararaman et al., 2020) aims to create embeddings for numbers, such that their cosine similarity reflects the actual distance on the number line. [9] The embedding should establish a correlation between the absolute distance of two numbers and the cosine similarity of their embeddings, ensuring that one increases monotonically with the other. To make the embedding contextualize for evaluation tasks, a bi-LSTM with soft attention is used to predict a bucket for each number, using the Numeracy-600K dataset (Chen et al., 2019a). The optimization objective of DICE enhances the performance of word embeddings on numeracy and magnitude tasks as well as probing tasks like list maximum, decoding, and addition.

Embedding (2): Due to the infinite variety of numbers, their appearances in textual corpora are extremely rare, leading to *Out-Of-Vocabulary (OOV)* representations for the majority. Jiang et al. (2020) delve into the numerical statistics within existing corpora, highlighting that the limited vocabulary size of traditional embedding methods contributes to the failure of these models. For example, while 6.15 percent of all unique tokens in English Wikipedia are numbers, the GloVe embedding, which is partially trained on Wikipedia, contains only 3.79 percent of them. To handle

[9] Number line is a line on which numbers are marked at intervals.

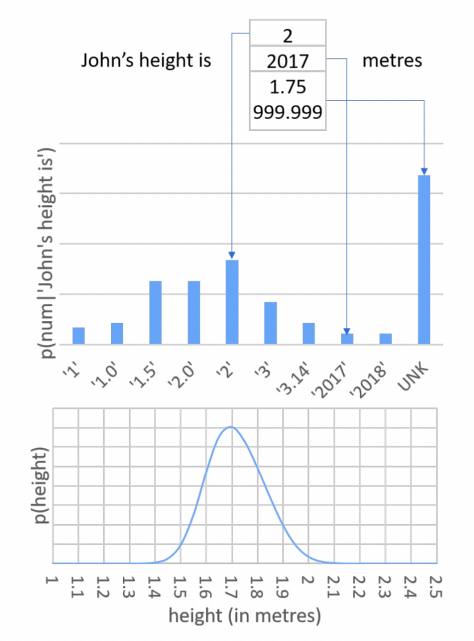


Figure 3.10: Modeling quantities with a categorical distribution over a fixed vocabulary does not reflect the smoothness of the continuous distribution. Image is taken from Spithourakis and Riedel (2018).

the OOV problem, an embedding of a quantity is defined as a weighted average of the similarity to prototype embeddings. Prototypes are typical numbers that are induced from the training set using self-organized mapping (Kohonen, 1998) or a Gaussian mixture model. Self-organized mapping is a clustering method and in a Gaussian mixture, each distribution is like a cluster. Each cluster centroid will be a prototype, which will have an embedding assigned to it. The embedding of a number is then calculated as the weighted average of these prototype embeddings. These weights are determined by a similarity function between the number and a prototype, such as the absolute difference between the prototype and the number. Consequently, the infinite number of embeddings for numbers is effectively condensed into a set of predefined prototypes, where the embedding of each number is derived from its distance to these prototype embeddings. The proposed method improves the performance of number-related intrinsic and extrinsic tasks. Nevertheless, these embeddings are learned with a skip-gram model and are not contextualized, e.g., "2019" as a "digit" and "year" have the same representation.

3.3.3 Contextualized Representations

Contextualized word embeddings are often generated using language models. In this section, we review related work in this area. Most methods focus on modifying the masked language modeling or next-word prediction task into a number prediction task. This is typically done through regression, digit prediction, or distribution prediction.

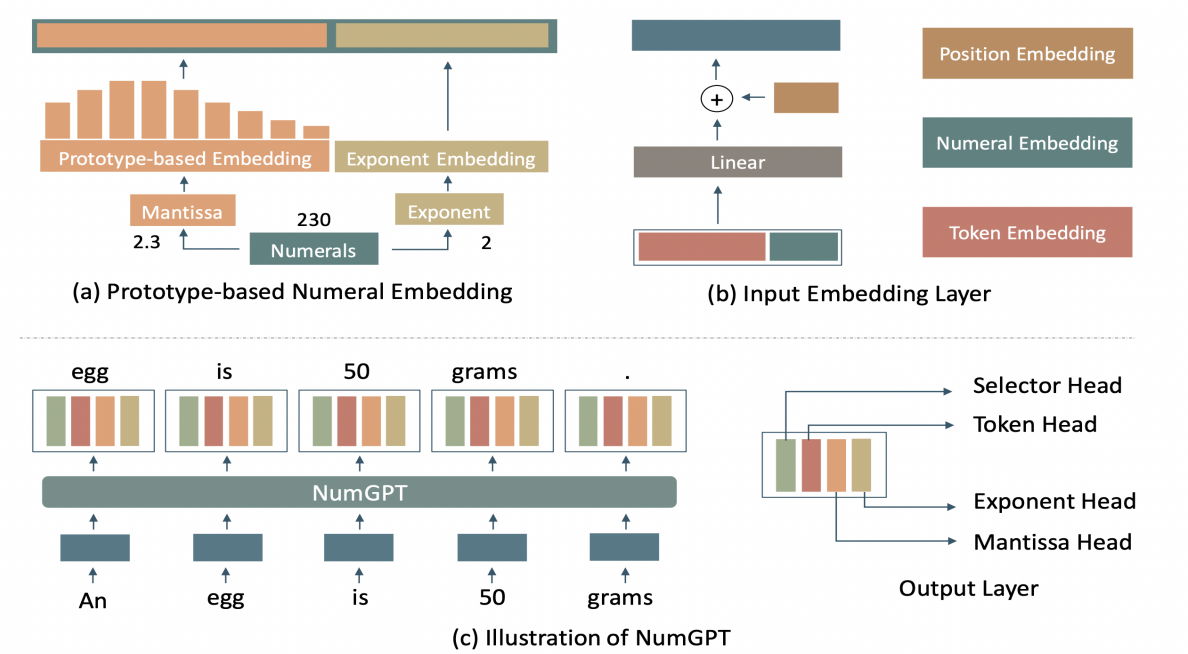


Figure 3.11: The model architecture of NumGPT from Jin et al. (2021): (a) Shows the prototype-based embedding to encode the mantissa and another embedding to encode the exponent. (b) The numeral embedding and token embedding are fused together. (c) NumGPT has four heads. If the *Selector Head* predicts a token, then the *Token Head* outputs a token. Otherwise, *Mantissa Head* and *Exponent Head* output a number.

Embedding (3): The first work to focus on numerical language modeling is the work of Spithourakis et al. (2016). Issues with language models arise when numerals are treated similarly to other words, using categorical distributions that ignore the smoothness of continuous attributes. An example of this problem is illustrated in Figure 3.10, where not only there are two distinct representations for the value "2", but also the majority of values is grouped under the unknown token "UNK". The authors explore various strategies for modeling quantities in an RNN-based language model, noting that quantities need to be handled differently from textual tokens. Predicting numerical values is done in three ways: *softmax* variants, a *digit RNN model*, and a *mixture of Gaussians*. The proposed method improves on perplexity in clinical and scientific texts and is the basis for later work in text completion on clinical text (Spithourakis et al., 2016). In the following, we describe the variants:

Softmax model: In this model, the assumption is that quantities come from a finite vocabulary, and dedicated out-of-vocabulary tokens for numbers are defined. The input token embedding for numbers is generated with character-based embeddings consisting of digits (0-9), the decimal point, and an end-of-sequence character. A hierarchical softmax based on the token class (number or word) predicts the next token and decouples quantities from words.

Digit RNN model: In this approach, a digit-by-digit prediction task is employed, estimating the probability of a number based on the probabilities of its digits using an RNN for character-based prediction. A key advantage of this method is its ability to handle an open vocabulary, thereby eliminating the need for unknown tokens.

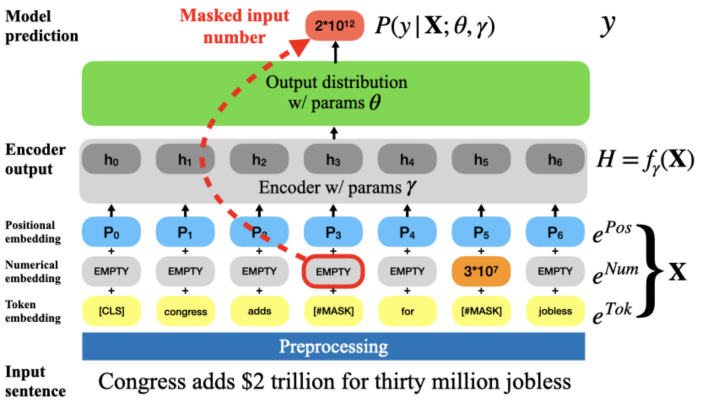


Figure 3.12: Overview of the model architecture from Berg-Kirkpatrick and Spokoyny (2020), comprising a sentence representation X fed into the encoder with parameters λ, and an output distribution across the real number line with parameters θ. A numerical embedding is added to enhance numerical prediction; Image taken from Berg-Kirkpatrick and Spokoyny (2020).

Mixture of Gaussians model: The probability of a number in text is computed using a probability density function over all real numbers, estimated via a mixture of Gaussians. However, since the probability of a continuous random variable that equals an exact value is always zero, the authors resort to using the probability mass function for a discrete approximation.

Embedding (4): NumGPT (Jin et al., 2021) integrates a specifically designed quantity representation and loss function to the auto-regressive language model of GPT (Radford et al., 2018). Figure 3.11 illustrates the architecture of NumGPT. Inspired by previous work (Jiang et al., 2020; Sundararaman et al., 2020; Zhang et al., 2020), a hybrid embedding that captures scale as well as precision is designed. Numbers are converted into their scientific notation, consisting of an exponent and a mantissa. The embedding of a number is then the concatenation of the exponent and mantissa embeddings. The exponent uses a fixed vocabulary ranging from -10 to 10 and for the mantissa, the prototype approach from the DICE embeddings is adapted (Jiang et al., 2020; Sundararaman et al., 2020). The language modeling objective of GPT is modified to first predict whether a token is textual or numerical. If a token is textual, the standard cross-entropy loss is computed. If the token is a number, the cross-entropy loss is applied separately for the prediction of the mantissa and the exponent. Figure 3.11 (b) shows how token and numeral embeddings are concatenated and combined with positional embeddings. NumGPT is trained on Wikipedia articles and tested against the base GPT architecture on a variety of synthetic tasks. The results show that a dedicated numerical embedding can boost a language model's performance on number comparison, magnitude understanding, and solving math world problems.

Embedding (5): Berg-Kirkpatrick and Spokoyny (2020) focus on contextualized number prediction, where a real numeric value is predicted in a masked language modeling style. Two sets of experiments are conducted: masked number prediction and numerical anomaly detection. In masked number

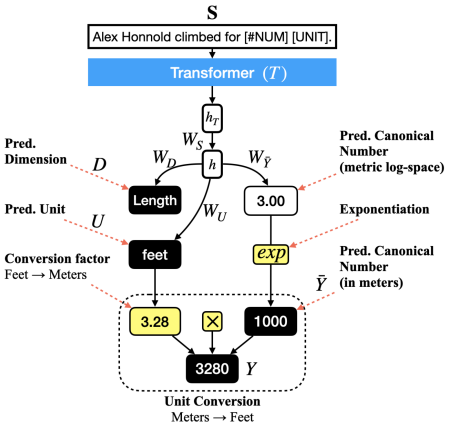


Figure 3.13: An example for inference in the GeMM model, where the fixed operations used during unit conversion are shown in yellow and different components of model prediction in black; Image taken from Spokoyny et al. (2022).

prediction, the goal is to predict the masked number token, and in numerical anomaly detection, the model should detect if a specific number in the sentence is anomalous. In addition to the change in the pre-training objective, the authors propose an architectural change in the embedding layer of language models. Figure 3.12 shows the new construction and masked number prediction task, where in addition to the token and positional embeddings, a numeric value embedding, e^{NUM}, is combined to learn a better representation for numbers. The construction of e^{NUM} involves either employing character-based RNNs on the scientific notation of the numbers (e.g., "d.ddde+d") or encoding solely the 10-base exponents, indicating the magnitude of the values. The new token embeddings are used in a bidirectional RNN or a transformer-based language model for training on masked number prediction. The model's output consists of parameters defining a value distribution based on the context of the sentence. Regarding the distribution type, the author conducted experiments using a range of methods, including Log Laplace, flow-transformed Laplace, discrete latent exponent, and Gaussian mixture model.
The models are trained on a set of financial news articles and academic papers and evaluated on number prediction with log mean absolute error and accuracy of exponent prediction and also for anomaly detection. Based on the experiments, it is concluded that the transformer-based model outperforms the RNN language model and that using the base-10 exponent embedding is superior to character-based RNNs for number encoding.

Embedding (6): Spokoyny et al. (2022) extend the masked number prediction from Berg-Kirkpatrick and Spokoyny (2020) to consider units, and introduce *Masked Measurement Prediction (MMP)*, where the model learns to reconstruct a number and a unit given a masked text. The model that is proposed to perform MMP is named the *Generative Masked Measurement model (GeMM)* and is shown with an example in Figure 3.13. GeMM emphasizes the importance of units in predicting the correct number. For example, given the question "How long did Alex Honnold climb for?"

a single number as an answer is meaningless, and both answers "100 meters" or "4 hours" could be valid. Moreover, measurements carry intricate semantic meanings influenced by standards and natural phenomena. Taking a text on rainfall as an example, units like "inches per year (in/y)" and "meters per second (m/s)" share the dimension of velocity. However, the choice of "in/y" indicates a focus on total regional rainfall, while "m/s" suggests an examination of rain droplet speed.
The proposed generative process samples a discrete *dimension* variable conditioned on the input sentence, which is used to sample a discrete unit variable compatible with the dimension. For example, conditioned on the predicted dimension of "velocity", the GeMM model will output a distribution over the units of "velocity" such as "miles per hour", "meters per second", and "inches per year". The dimension is either selected from a set of pre-defined units or treated as a latent vector, which is learned during training. Separately, a value for the measure is predicted conditioned on the sentence and the predicted unit.
GeMM is based on a transformer-based language model and is trained to predict a distribution over values (log Laplace), aligning with the approach of Berg-Kirkpatrick and Spokoyny (2020). The study demonstrates enhancements in both unit and number prediction through conditional generation, emphasizing the crucial role of units for improved numerical representation.

Embedding (7): Petrak et al. (2023) introduce another pre-training approach for enhancing the numeracy of language models on various numeracy-dependent downstream tasks. In contrast to other numerical language models discussed here, Petrak et al. (2023) aim for a pre-training paradigm that does not change the architecture or require dedicated numerical embeddings. However, the study focuses on sequence-to-sequence models such as T5 (Raffel et al., 2020).
The proposed approach is called *arithmetic-based pre-training* and consists of a constructive loss that combines subword-level and character-level tokenizations and a denoising objective called *inferable number prediction* task, to improve the model's capability of working with numbers. Inferable number prediction is inspired by previous work on domain adaptive pre-training (Gururangan et al., 2020), where an additional pre-training step is used to bridge the gap between the vocabulary of the original model and the new domain.
The contrastive learning utilizes the multiple negative ranking loss (Henderson et al., 2017) to improve the representation of numbers. For example, the model should learn a similar representation for the number "1108.89", whether it is tokenized as characters ("[1, 0, 8, ., 8, 9]") or sub-words ("[10, 8, ., 89]"). If a number is common in the pretraining corpus, subword-based encoding may provide more information. Otherwise, character-level tokenization could yield better insights. For contrastive loss, all numbers in the batch are considered independently of the input sequences. Each number is tokenized twice: once at the character level and once using subword-based tokenization. Character-level tokenization serves as the anchor, while the subword-based tokenization of the same number acts as the positive sample. All other numbers in the batch are treated as negative samples for the anchor.

The inferable number prediction task is a variant of masked language modeling, in which two sentences, one with a masked number and one without masking, are presented to the model and the model should reconstruct the masked number using the context of both sentences. This input is dependent on the downstream task, e.g., for table-to-text generation one sentence is the linearised form of the table and the other is the description with a masked number.
After the arithmetic-based pre-training, the model is tested on reading comprehension and text-to-table tasks and shows improvement over baselines.

3.3.4 Language Models in Finance

Similar to many domain-specific tasks, language models trained on general domain data are not suitable for financial settings, where the text contains a specific vocabulary and numbers.

Embedding (8): FinBERT (Araci, 2019) is a BERT-based language model for financial NLP tasks and is mainly evaluated for sentiment analysis. To this end, the author introduces a new dataset, TRC2-financial, consisting of financial news articles from Reuters. Domain-specific pre-training is then done by additional fine-tuning on the TRC2 dataset or training sentences from a sentiment classification dataset. However, the authors do not investigate the effect of pre-training for quantities in text or perform any numerical reasoning tasks.

Embedding (9): FiNER (Loukas et al., 2022) performs financial and numeric entity recognition or XBRL (eXtensive Business Reporting Language) tagging in reports from publicly traded companies. XBRL is an XML-based language for financial documents, where most tagged tokens are numeric. It is mainly used in periodic financial reports in the US, UK, and EU and contains a larger set of entity types in comparison to normally named entity recognition (6k in full XBRL, where 139 are considered by the authors). They provide a novel dataset from 10k annual and quarterly English reports with 1.8M total sentences. Since the majority of tags contain numerical information, the authors look deeper into the tokenization and the representation of numerical values in BERT. The finite vocabulary of BERT is insufficient for numeric expressions, e.g., the BERT tokenizer splits everything on punctuation, therefore, the token "9,323.0" is split into five subword units, [9, ##, ##323, ##., ##0]. This excessive fragmentation harms the performance of such models, as the probability of producing nonsensical sequences of labels increases. To overcome this shortcoming, the authors propose two methods: *BERT + [NUM]* and *BERT + [SHAPE]*.
In *BERT + [NUM]*, numbers are detected using regular expression and replaced with the [NUM] token, for a uniform representation. The [NUM] pseudo-token is learned during fine-tuning and avoids fragmentation of numerals.
BERT + [SHAPE] replaces numbers with pseudo-tokens representing the shape of a number, e.g.,

"53.2" becomes "[XX.X]", and "40,200.5" becomes "[XX,XXX.X]". 214 tokens for different magnitudes are considered, and their representations are learned respectively during fine-tuning.
The numeric pseudo-tokens improve upon the task of XBRL tagging in comparison to the vanilla BERT model or FinBERT. The error analysis reveals that the model fails to fully comprehend highly technical details, for example, it fails to distinguish between different expense types. Moreover, temporal information is another bottleneck and is often misclassified.

After examining various numerical embeddings, one might wonder if they are suitable for retrieval use cases. Non-contextual embeddings provide static representations, ignoring context information. While such embeddings were used in older retrieval models, they are no longer ideal for document or query representation.
Contextual models consider the surrounding context but come with their own set of issues. Typically, their training objectives are limited to predicting numbers or units, overlooking other textual elements. It is surprising that no study ensures that textual element representations remain unaffected by numerical fine-tuning. Almost all models require pre-training from scratch, but do not share their pre-trained checkpoint or training code. [10] Except Berg-Kirkpatrick and Spokoyny (2020) all other models consider numbers in isolation and disregard unit information. Although Berg-Kirkpatrick and Spokoyny (2020) emphasize units, their training objective completely ignores textual tokens and focuses only on a predefined set of units, limiting the model's capabilities. In summary, while these methods are effective for specific domains and tasks, they are not directly suited for IR settings.
Similar arguments apply to financial language models. FinBERT is limited to the financial domain, whereas, in this thesis, we aim to create a general-purpose retrieval system. It is worth noting that we experimented with the *BERT+[SHAPE]* representation for our quantity-aware models, but it did not enhance performance and was subsequently disregarded.

3.4 Reading Comprehension and Numerical Reasoning

Another relevant domain for quantity understanding is question answering and numerical reasoning. Unlike embedding and representation techniques, the focus is no longer on tweaking the capabilities of embeddings on numerical probing tasks but more on a specific downstream task of reasoning on a passage or table that contains quantities.
Question answering systems in this domain are typically not open-domain, meaning relevant passages or tables are provided without needing an initial retrieval stage. However, in some cases, the model must answer based on the accumulated world knowledge encoded in its parameters. In real-world scenarios, these systems are often paired with a retrieval component to first gather relevant

[10] Only Jin et al. (2021) provided their code upon request but the resource for training such a model from scratch is still needed.

passages from a large corpus. A reading comprehension or reasoning module then condenses that information to answer the given questions.

Unlike quantity-aware retrieval, passage and table comprehension are more prominent in the literature, and due to the large number of works in this domain, the list presented here is by no means comprehensive. Here, we discuss the methodology of a few selected works to provide an overview.

With recent advances in large language models [11] efforts to create dedicated models for numerical reading comprehension have decreased and the focus has shifted to solving such problems with *Chain-of-thought* reasoning (Wei et al., 2022), Scrachpads (Nye et al., 2021) or equipping large language models with calculators, tools or external interpreters (Gou et al., 2023; Drori et al., 2021). The survey by Lu et al. (2023) provide a good overview of current research in this domain.

Table 3.6 contains a list of approaches discussed and the type of task they refer to. The numbering is aligned with the methods in the text. First, we provide a brief history of this task and introduce the datasets used in this domain. Then, we present an overview of the methods listed in Table 3.6.

Table 3.6: List of works investigating numerical reasoning for reading comprehension or entailment tasks.

Model	Approach	Task
1. Dua et al. (2019)	output layer to support the four different kinds of answers: (1) question span (2) passage span (3) count (4) arithmetic expression	reading comprehension
2. Ran et al. (2019)	adds a reasoning module as a graph neural network: nodes are numbers in text and edges are relations between them (>, <=)	reading comprehension
3. Chen et al. (2020b)	adds entities and their relations to numbers to NumNet graph and question representation to graph reasoning	reading comprehension
4. Chen et al. (2020c)	language model reader to encode passages LSTM-based programmer to generate reasoning programs	reading comprehension
5. Geva et al. (2020)	adding two-step pre-training with numerical data (12+2=14) or textual data (passage+quersion+answer) generated with templates	reading comprehension
6. Wei et al. (2022)		reading comprehension
7. Wang et al. (2024)	generate a reasoning process by decomposing the input, using synthetic data for training	table & text comprehension

With the release of the DROP dataset (Dua et al., 2019) for numerical reading comprehension, numerical reasoning, and understanding found their way into the question answering realm. The questions are designed to encourage systems to learn addition, subtraction, comparison of values, and other arithmetic tasks. Another widely used dataset is NQuAD (Chen et al., 2021) with more than 70,000 questions, which takes a slightly different approach to question generation. The dataset provides fine-grained multi-option questions from news articles, and a given system should predict

[11] It is worth noting that at the time of writing this thesis, the models specifically designed for numerical question answering still perform better than state-of-art large language model models such as GPT-4 and Gemini (Team et al., 2023; OpenAI et al., 2023).

the correct option given the article as context. The answers do not need numerical reasoning or arithmetic but rather an understanding of the relation of numbers to other entities. An example of a question from NQuAD is "Driven by self-discipline, the five major banks new mortgage interest rates are approaching nearly ___ %", where the system is presented with four options to fill the blank, e.g., "(A) 0.04 (B) 1.986 (C) 2 (D) 2.5". In this case, the system should comprehend the paragraph as context and notice that (C) is the correct answer.

Although in this section we primarily focus on numerical reasoning through reading comprehension, numerical reasoning encompasses various tasks, such as solving math word problems. However, since these other tasks are not directly relevant to retrieval, we do not discuss them here. For instance, solving arithmetic problems demonstrates numeracy capabilities but is not typically integrated with a retrieval system and cannot serve as a downstream application for our quantity-aware retrieval models. Furthermore, our quantity-aware IR systems require an understanding of scales and numerical comparisons rather than advanced mathematics. While reading comprehension datasets often include questions testing for scale understanding, this is not the case for many other numerical reasoning datasets, making them less relevant to our overall task.

We briefly describe these other tasks but do not present specific models used in these tasks. LĪLA (Mishra et al., 2022a) and NumGLUE (Mishra et al., 2022b) have comprehensive benchmarks for numerical reasoning, mainly for testing large language models. NumGLUE is a multi-task benchmark comprising 8 different tasks, which can involve world problems, reasoning strategies like commonsense reasoning or reading comprehension combined with simple arithmetic. The set of 8 tasks proposed by NumGLUE are as follows:

- *Commonsense + Arithmetic Reasoning*: Questions that require word knowledge and simple arithmetic, e.g., "How many faces do 10 dice have?".

- *Domain Specific + Arithmetic Reasoning*: Questions that require knowledge in a specific domain and simple arithmetic, e.g., "How many units of hydrogen are required to produce 10 units of water?".

- *Commonsense + Quantitative Comparison*: To answer these types of questions, the model should have world knowledge and be able to perform numerical comparisons, e.g., "A golf ball weighs 40g and a baseball weighs 150g. Which has a higher gravitational force?'.

- *Arithmetic Word Problems*: This task includes single- and multi-step word problems with addition, multiplication, subtraction, division, and other math topics, e.g., "John had 5 apples. He gave 3 to Peter. How many apples does John have now?".

- *Fill-in-the-blanks Format* In this case, the questions from Arithmetic Word Problems are converted to fill-in-the-blanks, e.g., "John had 5 apples. He gave 3 to Peter. John has ___ apples.".

- *Reading Comprehension + Explicit Numerical Reasoning*: This is a subset of the DROP dataset, where the answers are numeric values.
- *Reading Comprehension + Implicit Numerical Reasoning*: Another subset of the DROP dataset, where the answer is a span of text.
- *Quantitative NLI*: Problems that require simple arithmetic calculations to be performed to accurately classify the relationship between the provided premise and the hypothesis, e.g., "Premise: John had 5 apples. He gave 3 apples to Peter. Hypothesis: John has 2 apples now. Does the hypothesis entail, contradict, or is neutral to the premise?".

They show that tasks involving numerical understanding are challenging for large language models, obtaining poor scores not only in zero or few shot settings but also after fine-tuning. Moreover, numerical tasks that contain common sense knowledge are the hardest to solve for the models, with better performance observed in answering span-based questions compared to numeric answers. Additionally, the research points to the potential improvement in model performance on common sense knowledge tasks through numerical information retrieval, emphasizing the broader relevance of our work in this thesis beyond the IR domain. LĪLA, which is created by the same authors, is a unified mathematical reasoning benchmark that consists of 23 mathematical reasoning tasks. LĪLA extends all 23 datasets to include a solution program in Python as opposed to only an answer, and instruction annotations. The programs serve as reasoning chains for each question in the benchmark. As mentioned before, many of the math-related tasks are focused on solving equations or arithmetic problems, which is not related to the topic of this thesis.

Reasoning (1): Alongside the DROP dataset (Dua et al., 2019), the NAQANet model was proposed for neural reading comprehension with symbolic reasoning. NAQANet can answer three types of questions: spans, counts, and addition or subtraction over numbers. Similar to a previous QANet model (Yu et al., 2018), the architecture is composed of embedding, encoding, passage-question attention, and output layers. The output layer is altered to support four different kinds of answers:

- *(1) Passage span* and *(2) question span* type, predict an answer span in a passage or question.
- *(3) Count* accounts for counting as a multi-class classification problem between 0 to 9.
- *(4) Arithmetic expression* locates multiple numbers in the passage and finds the answer by adding or subtracting them. To model this process, all numbers are extracted and a minus one, plus one, or zero is multiplied by them before summing up to evaluate the final answer.

NAQANet outperforms models without a numerical reasoning module but still fails at complex tasks. Moreover, as mentioned in Section 3.3.3, training NAQANet for question answering indirectly encodes some numerical understanding of scale and comparison into token embeddings. We

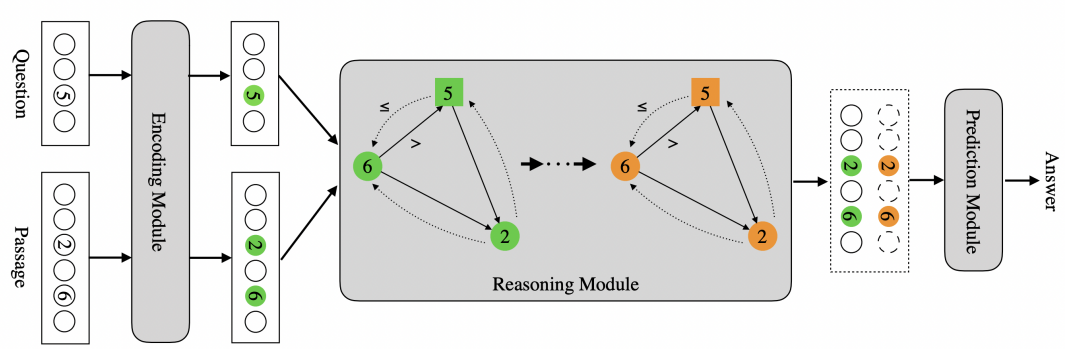


Figure 3.14: The model architecture of NumNet from Ran et al. (2019). The model consists of an encoding module, a reasoning module, and a prediction module. The graph encodes the numerical relations and it is leveraged by a graph neural network for reasoning. For example, the edge from "6" to "5" indicates that "6" is greater than "5".

take advantage of this finding in Chapter 5, when we propose task-specific fine-tuning to enhance quantity understanding in IR models.

Reasoning (2): NumNet (Ran et al., 2019) is another framework, which consists of an encoding module, a reasoning module ,and a prediction module, where the encoding module and prediction module are the same as the NAQANet. The main contribution of NumNet lies in the reasoning module, which leverages a *Numerical Graph Neural Network (NumGNN)* between the encoding and prediction modules to perform numerical reasoning. Nodes of the graph are numbers in questions and passages, and edges encode numerical relationships among numbers. There are two types of relationship or edge types for *greater than* and *lower than or equal*. Reasoning is done by message propagation between each node to its neighbors, using relation-specific transform matrices. An example extraction of a given graph is visualized in Figure 3.14. With this numerically aware graph, the authors manage to capture numerical conditions in text. One limitation of NumNet is its inability to handle cases where an intermediate number must be derived (e.g., from an arithmetic operation), as the predefined graph cannot accommodate new node additions. Additionally, the method does not differentiate between various number types or their relationships to entities in the text. Moreover, the reasoning module ignores the question text, thereby omitting crucial information.

Reasoning (3): Shortly after the success of NumNet, QDGAT (Chen et al., 2020b) set out to fix the shortcomings of NumNet. Two main issues were the lack of understanding of different number types and their connections to entities, as well as the omission of question representation in numerical reasoning. To address the first issue, they constructed a heterogeneous directed graph, where nodes represent entities and various types of numbers, and edges represent different types of relationships. To tackle the second issue, a contextual encoding of the question is added to the graph reasoning process to identify important numbers. QDGAT remains state-of-the-art to this date, showcasing that the separation of quantity information from the textual content is somewhat

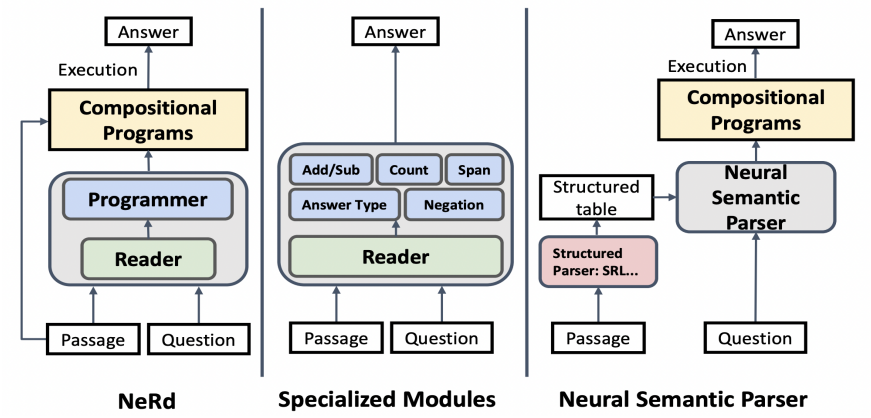


Figure 3.15: Comparison reading comprehension methods, where NeRd is an instance of the neuro-symbolic family. The components in grey boxes are the neural architectures. Specialized Modules: are usually pre-trained language models with specialized modules for each type of question. These models are more rigid and less scalable to new domains and cannot be extended to multi-step complex reasoning; Neural Semantic Parser: apply semantic parsers to the structured representation of the passage, which suffers from a cascade of error. NeRd is domain-agnostic and easily adaptable. It includes a reader, e.g., BERT, and a programmer, e.g., LSTM, to generate programs. Image taken from Chen et al. (2020c).

necessary to comprehend complex quantity relationships in text. Even language models like GPT-4 are still struggling on this task (OpenAI et al., 2023).

Reasoning (4): Chen et al. (2020c) introduce a neuro-symbolic reader for numerical reasoning tasks (NeRd). A passage is encoded by a reader module, which is a language model like BERT, and fed into a programmer, e.g., LSTM, to generate a program for multi-step reasoning. The programmer takes embeddings from the reader as input and then decodes a program as a sequence of tokens. The output after execution of the generated program provides the answer to a question. They introduce a domain-specific language that is used to interpret tokens generated by the programmer. The language includes operators that perform arithmetics, counting, and sorting and also specific tokens for selecting spans or numbers from the passage and question. Similarly Saha et al. (2022) also use neuro-symbolic module networks (Gupta et al., 2020) to parse the query into a specialized program and execute it step-wise over learnable reasoning modules. Since the program is a sequence of tokens generated based on the context and the questions, the neuro-symbolic models are quite flexible in the variety of questions they can solve. However, optimizing over the large space of possible solutions is usually the main challenge of these models, which is often solved with some type of pruning or distant supervision. Figure 3.15 compares the neuro-symbolic reasoning to other approaches, where NumNet and NAQANet fall into the specialized modules category. The compositional nature of neuro-symbolic models allows for more scalability, whereas for each new operation, a new dedicated module has to be added to the specialized module models.

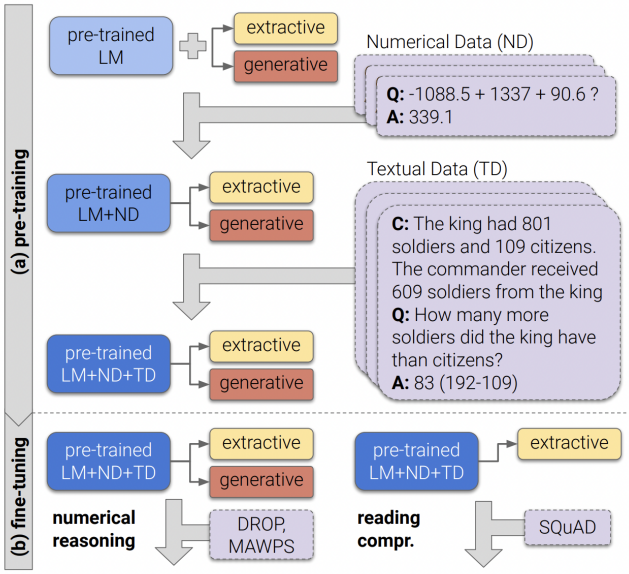


Figure 3.16: Figure from Geva et al. (2020), showing the two pre-training strategies for GENBERT: (a) two pre-training steps with synthetic numerical data (ND) and textual data (TD); (b) fine-tuning the model over either numerical reasoning datasets or reading comprehension.

Reasoning (5): Geva et al. (2020) take a different approach, and instead of focusing on architectural changes, they inject numerical reasoning into GENBERT by adding two additional pre-training steps. In GENERT an answer can either be an extracted question and passage or generated from a decoder. The wordpiece tokenization of BERT is modified to digit tokenization for numbers. Additionally, a random shift is introduced to the position embeddings to prevent overfitting on short inputs by randomly shifting all position IDs by an integer.

The first pre-training step introduced by Geva et al. (2020) is on numerical data in the form of mathematical operations, e.g., "$43 - 4 + 11 = 50$". The type of data is easy to generate, since it does not involve textual content. The authors claim that this type of data teaches the model to compute the value of numbers from their tokens and to perform numerical operations.

The second step of pre-training is to automatically generate question and answer pairs, in order to enhance the model's ability to understand numerical reasoning expressed in natural language. To this end, the authors generate templates following the approach of Hosseini et al. (2014) for math word problems. The templates contain abstract tokens that can be replaced with entities and numbers and are used to generate sentences, which are utilized to create questions based on them. Figure 3.16 demonstrates the two pre-training steps. The model is then fine-tuned on specific tasks of reading comprehension or numerical reasoning. The study shows that the task-specific pre-training enhances the model's understanding of quantities.

We take a similar approach as Geva et al. (2020) in Chapter 5 and use template generation the create synthetic data for fine-tuning a quantity-aware retrieval system.

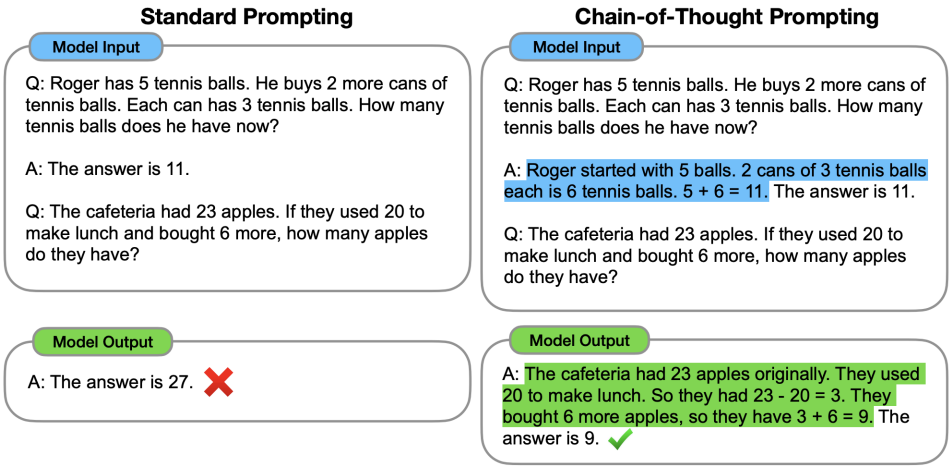


Figure 3.17: An example of demonstrating the difference between normal prompting and chain-of-thought prompting. Image is taken from Wei et al. (2022).

```
Input:
2 9 + 5 7

Target:
<scratch>
2 9 + 5 7 ,  C: 0
2 + 5 , 6 C: 1  # added 9 + 7 = 6 carry 1
, 8 6 C: 0  # added 2 + 5 + 1 = 8 carry 0
0 8 6
</scratch>
8 6
```

Figure 3.18: Example of input and target for addition with a scratchpad. The carry is recorded in the digit following "C:". Comments (marked by #) are added for clarity and are not part of the target. Image is taken from Nye et al. (2021).

Reasoning (6): Chain-of-thought prompting (Wei et al., 2022) is an approach designed to align large language models more closely with human thinking by breaking down complex problems into logical, sequential steps. The proposed approach augments each exemplar in few-shot prompting with a chain of thought for an associated answer. This approach enables large language models to decompose multi-step problems into intermediate steps and provides an interpretable insight into the behavior of the model. The authors primarily concentrate on math word problems and demonstrate that chain-of-thought prompting, when employed with 540B parameter language models, performs comparably with task-specific fine-tuned models. It is crucial to note that the language model needs to be sufficiently large for these advantages to manifest. An example of the standard prompting is changed into a more elaborate chain-of-thought is shown in Figure 3.17.

Scrachpads (Nye et al., 2021) introduced another technique similar to chain-of-thought prompting. In both cases, intermediate reasoning steps are added to enhance performance. In this case, intermediate steps of an algorithm are encoded as text and the model is trained to emit them to a buffer called *scratchpad*. An example is shown in Figure 3.18 for the algorithmic induction task of learning long

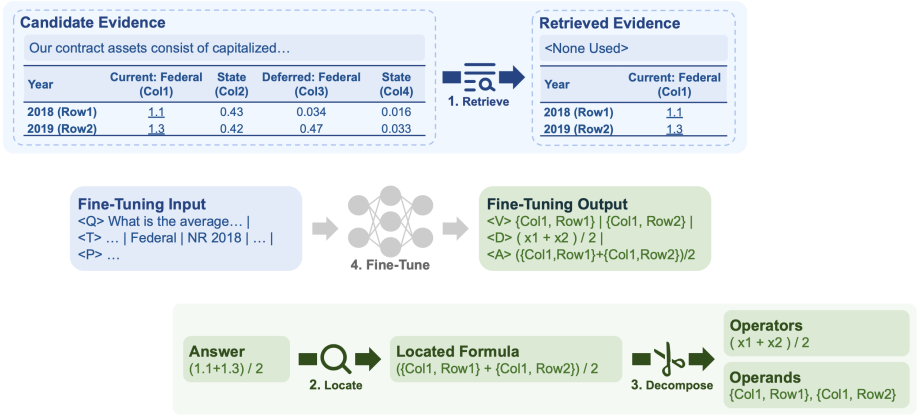


Figure 3.19: Illustration of ENCORE for a single example, consisting of four steps: 1. Retrieve question-related evidence. 2. Locate the table heads of each value in the formula. 3. Decompose the located formula into operators and operands. 4. Fine-tune the model with the input and the generated output. Image is taken from Wang et al. (2024).

addition. To teach a model to add 29 to 57, the steps of the grade-school long addition algorithm are written out explicitly.

Reasoning (7): Wang et al. (2024) aim to enhance the numerical reasoning in large language models for questions answered on text and tables. Their study introduces ENCORE, which generates a reasoning process decomposed from answers to avoid irrelevant evidence. The reasoning generation is done in four steps, as shown in Figure 3.19. First, a retriever finds the relevant evidence for the given question. Each text paragraph and table column is concatenated with the question and is fed to a binary classification model to generate correlation confidence. The top-k text paragraphs and table columns with the highest correlation confidence are chosen as evidence. The second step is designed to reduce the difficulty of value and table understanding by changing the value format in answers. The values in the answer are substituted by locating their respective headers in the table. The third step is to decompose the answer to reduce the complexity of reasoning. The formula for generating the answer is decomposed into operators and operands. The constructed formulas and answers from the dataset are used as targets and the question and the retrieved evidence as input to fine-tune a sequence to sequence model.

Even with the reasoning process generated by ENCORE, the model could still struggle to learn how to produce such processes because of the limit of the training data. To aid the model in learning to generate the reasoning process, the authors synthesize questions, answers, and reasoning processes based on different templates, and then pre-train the model with all these data as the multi-task training. The synthetic data include data for:

- *Table Location Prediction*: Given the row and column headers of one cell, the model should predict the value of this cell.

- *Table Calculation Prediction*: Given the column and the calculation type, the model should generate the correct operator and operands.
- *Hierarchical Table Prediction*: For better operand extraction for hierarchical table structure with multi-level headers, the model should predict the name of the first level of each given table header.

The study concludes that smaller-size language models, e.g., BART_{large}, fine-tuned with the reasoning process of ENCORE and pre-trained on synthetic data, outperform reasoning with much larger models, e.g., GPT-3.5-turbo. This is another successful instance of pre-training on tasks-specific synthetic data that enhances numerical reasoning, similar to the work of Geva et al. (2020).

3.5 Numerical Relation Extraction

In this final section, we examine numerical relation extraction. In Chapter 4, we introduced a quantity extractor and explored the relationship between quantities and other textual tokens. Consequently, numerical relation extraction is loosely related to quantity extraction. The majority of models for relation extraction are rule-based and rely on syntactic or dependency parsing.
Numerical relation extraction explores the relationship between quantities and other textual arguments. Relation extraction on its own is the task of detection and identification of semantic relationships between two or more arguments in a text, such as entities and attributes. A sub-field of relation extraction focuses on quantities in the context of numerical relations between an entity and a quantity. Relations can identify attributes of named entities, such as "height" or "weight" or describe more general concepts such as "inflation rate". Unlike general relation extraction, which can heavily benefit from the addition of knowledge bases, certain types of quantities pose a challenge for this approach. Quantity and entity relations have a far broader context than entity-to-entity relations and are loosely defined, e.g., the entity "Eiffel Tower" has many known relations in the knowledge base for height and its attributes but quantities such as the number of "visitors per day" are not stored in a knowledge base. Moreover, quantities can be expressed in terms of their relative value, e.g., "FTSE 100 closes down 82 points", where the actual value is not mentioned, making the definition of a relationship more difficult. Most approaches in this domain focus on a rule-based system that finds patterns for keywords and phrases to extract the relations.

One of the first attempts in this domain is *NumberRule* and *NumberTron* (Madaan et al., 2016). Both systems tend to extract a set of relations, where the second argument is a quantity with a given unit, and the first argument is an entity. NumberRule, as the name implies, is a generic rule-based and precision-focused system. It first creates a dependency parse of a sentence and applies named entity recognition to identify candidates for identifying relations. Then, the shortest path in the dependency parse tree between a candidate entity and a quantity defines a relation. However, Num-

berRule is not a learning system and does not go beyond given keywords and a set of rules. Therefore, the authors propose NumberTron, which utilizes a graphical model like MultiR (Hoffmann et al., 2011) for relation extraction. NumberTron is trained using a set of keyword features (similar to the keyword list from NumberRule) and number features (characterizing scale and number types). Due to a lack of annotations, the system is trained with distant supervision and a perceptron-like training algorithm on an unlabeled text corpus with seed patterns from a knowledge base.

The first open IE model specifically built to handle numerical relation extractions is BONIE (Saha et al., 2017). BONIE uses bootstrapping to learn dependency patterns that express numerical relations on a large corpus of unlabeled data. The algorithm starts the training phase with a set of six manually selected dependency patterns, which are matched against a corpus to generate the seed facts that are used for bootstrapping. BONIE finds sentences that contain all words in a seed fact and generates (sentence, fact) pairs. In addition to the matching of words, quantities between facts and a sentence should match. In this context, two quantities match if they have the same unit and only a small difference in value. A threshold defines the acceptable difference. From extracting the sentences more patterns are generated and added to rules for the final extraction system.

Xart (Berrahou et al., 2017) is also a system for the extraction of quantitative data from text, modeled as n-ary relations. They model each object as a symbolic argument and its features as quantitative arguments associated with their attributes, i.e., the numerical value and measurement unit. The model employs a domain ontology for specific concepts of a given application domain and a core ontology that explains the relations in a global setting. Then, given a terminological dictionary of concepts, Xart finds quantitative relations for these domain-specific concepts in the text.
Entities and their quantitative properties are part of knowledge bases, but as mentioned previously, not as well maintained as other relations. Ho et al. (2022) address the problem of missing quantity information in knowledge bases such as Wikidata. They propose an iterative method, by first extracting relations using OpenIE and tagging named entities and quantities in text. The output of the first stage is Qfacts, similar to the previous work of Ho et al. (2019). Guided by the knowledge base schema and its entities, they generate a query to select candidate facts from the first stage for which the context indicates the predicate of interest. For example, for the predicate "height", leveraging the knowledge base schema, the type of predicates, such as "buildings" and required units of quantities like "meter" are discovered. The Qfacts are filtered to match the predicates and to have reasonable values. After some consolidation steps, new facts are added to the knowledge base.

In this chapter, we reviewed related work on quantity extraction and retrieval, exploring domains relevant to quantity understanding and representation in text. The standalone extractors discussed in Section 3.1.1 serve as baselines for evaluating our quantity extractor proposed in Chapter 4. The numerical relation extractors in Section 3.5 inspired the incorporation of contextual information,

such as concepts, into our quantity extractor.
Additionally, we drew inspiration from numerical indices discussed in Section 3.2.2 to design one of the two quantity-aware retrieval models detailed in Chapter 5. The second quantity-aware retrieval model is based on recent findings in enhancing numerical reasoning with additional training steps on synthetic data, as discussed in Section3.4.

4 Quantity Extraction

> "Number is the ruler of forms and ideas, and the cause of gods and demons."
>
> *Pythagoras*

The main theme of this thesis revolves around quantities and their importance in texts for conveying factual and accurate information. Hence, it is important to establish a common definition and vocabulary for what we refer to as quantities, and to consider different types of representations. The first thing that comes to mind is numeric values. Yet, not all numbers are valid quantities. Quantity is an amount, value, or measurement that expresses a magnitude (how much), a multitude (how many), or a duration (how long). Hence, not every numeric in a text falls into the mentioned categories. For example, the terms "Apollo 2", "Roewe 550" or " zip code 69120" contain numeric values, but they do not represent a multitude or magnitude, and therefore, are not considered as quantities in this thesis.

Quantities appear in different forms with different complexities, making their identification a daunting task. Despite their significance, a unified definition and a comprehensive system for extracting them are not yet at hand. As mentioned in Chapter 3, there is very limited research that directly studies quantity extraction, and most of it focuses on physical and scientific domains. Typically, quantity extraction serves as a pre-processing step for larger systems that handle retrieval or textual entailment tasks. (Banerjee et al., 2009; Li et al., 2021; Maiya et al., 2015; Sarawagi and Chakrabarti, 2014). Consequently, the definition of quantity varies based on the downstream application, and the performance of an extractor is not evaluated separately. As a result, when in need of a quantity extractor, one has to resort to a number of open-source packages, which not only are limited in their capabilities but also do not have a performance guarantee.

Due to this variety in definitions, the contextual information that is considered by each system is reduced to the essentials and particular needs of the downstream task. Most systems limit their definition to value and unit pairs, where the unit is part of the metric system (Foppiano et al., 2019). Nonetheless, outside of scientific and physical domains, any noun phrase describing a value or counting an item is a potential unit, e.g., in "5 bananas", *bananas* is the unit of counting. Moreover, numbers and units alone provide limited information about the context in which the quantity is mentioned. A more comprehensive representation of quantities should also include their behavior

and associated concepts. For example, in the sentence "German DAX fell 2% and S&P 500 gained more than 2%", the value and unit pair ⟨*2, percentage*⟩ indicates two different quantities in association with two different concepts, "German DAX" and "S&P 500", with opposite behaviors, *decreasing* and *increasing*. These subtleties are not captured by simplified models that only consider values and units. In this chapter, we focus on a comprehensive definition of quantities and propose a framework to extract them properly from unstructured text.

Contributions. In this chapter, we make the following contributions.

1. Introduce a unified definition for the quantities in the text that goes beyond value and unit pair and takes contextualized information into account.

2. Propose a *Comprehensive Quantity Extraction (CQE)* framework for extracting quantitative information from sentences.

3. Introduce the first-ever benchmark dataset specifically designed for evaluating quantity extraction systems for financial news articles.

4. Evaluate the performance of our proposed method against other extractors for the proposed benchmark for detection and normalization of quantities in text.

5. Design a web-based interface for interactive exploration of quantities in a piece of text.

Structure. We first discuss how quantities are presented in text and examine their properties, which leads to a further distinction between different types. After establishing a common taxonomy and definition, we present the CQE framework in Section 4.3. CQE can extract standardized values, both physical and non-physical units, changes and trends of these values, and the concepts associated with them. In Section 4.4, a new benchmark dataset for quantity extraction, named NewsQuant, is introduced. NewsQuant is the first benchmark dataset for quantity extraction and is carefully selected from a diverse set of news articles in the categories of economics, sports, technology, cars, science, and companies. CQE is compared against other open-source extractors on the NewsQuant dataset for a quantitative evaluation. Finally, in Section 4.5, we present an explorative web-interface to inspect the output of CQE and view a summary of extracted quantities.

References. Parts of this chapter are based on the peer-reviewed publications:

Satya Almasian, Vivian Kazakova, Philip Göldner, and Michael Gertz. CQE: A Comprehensive Quantity Extractor. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 12845–12859. Association for Computational Linguistics, 2023. URL https://aclanthology.org/2023.emnlp-main.793

Satya Almasian, Alexander Kosnac, and Michael Gertz. QuantPlorer: Exploration of Quantities in Text. In *Advances in Information Retrieval - 46th European Conference on Information Retrieval, ECIR 2024, Glasgow, UK, March 24-28, 2024, Proceedings, Part V*, volume 14612 of *Lecture Notes in Computer Science*, pages 171–176. Springer, 2024b. URL https://doi.org/10.1007/978-3-031-56069-9_13

4.1 Quantity Types Based on Appearance in Text

On the highest level of abstraction, a quantity appears either implicitly, e.g., "a few books", or explicitly, e.g., "five books".
Implicit quantities have no numeric values associated with them but rather the amount is indirectly indicated through special keywords. We frequently refer to quantities in our speech by choosing appropriate verbs and prepositions, such as "a". Some languages like Arabic even have dedicated conjugation for groups of two and another for three or more. These keywords or numerical words are often language-dependent and an exact equivalent might not exist in all languages. Additionally, the meanings of some keywords can change over time. An example of such a keyword is "myriad", which originally refers to ten thousand, but now loosely refers to a very large quantity. Consider the following sentences:

- "I ate *an* apple." refers to one apple.
- "I carried *a few* apples in a box." refers to a small number of apples but more than one.
- "Sara bought a new *pair* of shoes." refers to two items.
- "In my *teenage* years, I played in a band." refers to the range of 12 to 19 years of age.
- "The vase is more than a *century* old." refers to 100 years.

In all these sentences the amount associated with the quantity is inferred indirectly through certain keywords and is usually a vague amount or range of values. Moreover, some attributes and adjectives can represent implicit quantities related to an entity or concept. If we consider "Peter" as a "tall" individual and talk about his height, the sentence "Peter is 1.4 meters" is odd and incorrect, meaning that with "tallness" comes an inherent quality of a certain height that is implicitly defined. Identifying these types of quantities and reasoning about their values falls in the area of linguistics, which is, however, outside of the scope of this thesis.

On the other hand, the quantities that interest us are *explicit quantities*, which are indicated by either a numeric or word form, allowing for the deduction of an exact amount or range from the mentions in the text. Based on how a value is presented in the text, explicit quantities can occur in three forms:

1. *Word form:* The value is spelled out using words. The word form of the number "120" is "one hundred twenty". Usually, numbers between zero to a hundred are spelled out. For example, "the race has been suspended, since *five horses* suffered fatal injuries". Exceptions are cheques and certain payment documents, where the amount has to be written in both word form and numeric form.

2. *Numeric form:* The value is presented using digits and numerics. For example, "Lyft said it would offer *307,700,00 shares* of its common stocks to the public". This is the most common format present in the text, where a value is represented by using combinations of ten number symbols.

3. *Mixed form:* Combining a numeric with a word indicating magnitude or a multitude of the value or a unit. For example, "Cambridge Analytica received nearly *$6 million* for services provided to President Trump's campaign." Half of the value is in digit format and the rest in word form.

Due to the variety of numeric types, prefix, and suffix symbols, extraction of values in any format is more intricate than one imagines. We discuss these difficulties in the upcoming sections. Despite this general categorization, quantities can be further divided by their different properties, such as units and related concepts. To make such distinctions we require a definition of what we consider a quantity and how it is related to other tokens in a document. In the following, we look at our proposed quantity model and its properties.

4.2 Contextual Quantity Model

A quantity is an amount, measure, or number that can be associated with attributes such as magnitude, size, extent, volume, area, and other measurable characteristics. There exist various quantity models in the literature, where each focuses on certain aspects relevant to a downstream task. Rijgersberg et al. (2013) describe a quantity model for recording observations of the physical world where each record should contain four essential elements:

1. Phenomenon (object or event being observed), e.g., "a table"

2. Quantity kind (an aspect of the phenomenon being measured such as length or weight), e.g., "the height of a table"

3. Unit of measurement, e.g., "meter"

4. Numerical value, e.g., "1.05"

Another system is introduced by Forbus (1993) to describe qualitative processes in which a quantity is a changing parameter of the object in the study. Each quantity consists of two parts, an *amount*

and a *derivative*, which are both numbers describing a change in a physical element. Roy et al. (2015) brought the Forbus quantity model into a computer science setting, by focusing on three attributes:

1. *Value:* A numeric value or a range, e.g., "100" is a single value and "(100,102)" is a range containing multiple values.
2. *Units:* A noun phrase that describes what the value is associated with, e.g., "hour".
3. *Change:* How the parameter is changing, e.g., "the Apple stock increased more than 2%.", shows an increasing value of more than a certain amount.

The model by Roy et al. (2015) is called *Quantity-Value* representation and is denoted by the triplet (u, v, c). Quantity-Value representation is used by Roy et al. (2015) for the task of quantity entailment. Ho et al. (2019) extend the definition of Quantity-Value representation with an entity to introduce quantity facts. A quantity fact is a triple of

1. *Entity:* The entity, to which the quantity is referring.
2. *Quantity:* A triplet $q = (v, u, res)$ consisting of a numerical value v, a unit u, and a value resolution (change) $res \in$ {exact, approximate, upper bound, lower bound, interval}.
3. *Context:* A bag of words describing the relation between entity and quantity.

In this thesis, we combine all the definitions from above to introduce a contextualized quantity representation, considering four important aspects of a quantity in a textual context. A contextual quantity contains a value and unit pair and the context information around it. The definitions of value, unit, and change are similar to the Quantity-Value model. In addition, we borrow the notion of phenomenon from Rijgersberg et al. (2013) and rephrase it as *concept*. The concept is any object, entity, or event that the value is referring to.
For all upcoming definitions and models, we assume that a collection of documents is given, where each document consists of a sequence of sentences. There are also other segmentations of document structure, for example into paragraphs, possible but for our quantity model we focus on sentences as extraction units. We assume for the remainder of this chapter that co-references have been resolved in a document, prior to quantity extraction and that sentence segmentation is performed. [1] Each sentence consists of tokens and possibly quantities, where the tokens considered in this framework are single words. For the remainder of the chapter, we use *tokens* and *words* interchangeably.

Definition 4.1. *(Contextualized Quantity Representation).*
A sentence $s = (T, Q)$ *is a sequence of tokens* $T = (t_1, ..., t_l)$ *and a sequence of quantities* $Q = (q_1, ..., q_k)$*, where each quantity* $q = (u, v)$ *consists of a unit* u *and a numeric value* v*.*

[1] Segmentation of domain-specific text has its own challenges, which we do not discuss here, see, e.g., (Aumiller et al., 2021).

A contextualized quantity is defined as the tuple $cq := (q, ch, cn)$*, consisting of quantity* q*, change* ch *and concept related to the quantity* cn*.*
The function $Extract(s) = CQ$ *defines a mapping from a sentence* s *to the list of contextual quantities contained in the sentence* $CQ_i = (cq_1, cq_2, ..., cq_o)$*. If there are no quantities in the sentences, then* $Extract(s) = \{\}$*.*

It should be emphasized that even though we formulate our definition with a sentence being the smallest computational unit, the same idea can be extended to a paragraph or even an entire document. It is worth noting that by extracting contextual quantities, we distinguish ourselves from all previous work that considers only units and values for extraction.
In the following subsections, we look at each property of the contextualized quantity individually.

4.2.1 Values

A value is a real number or a range of values describing the actual amount, magnitude, multitude, or duration of a property. A value is represented by either a real number $v \in \mathbb{R}$ or the lower and upper bound of a range $[v_l, v_u]$ with $v_l, v_u \in \mathbb{R}$. For example, in the sentence "the car accelerates from 0 to 72 km/h", the value is the range "$v = [0, 72]$". In the sentence "the car accelerated to 72 km/h", the value is a single number "$v = 72$". The value is the primary attribute that distinguishes quantities from other tokens in the text, providing them with scale and enabling comparison. Furthermore, a value cannot be mapped to a discrete space alone and it requires a continuous one. While it is typically represented numerically, it can also be expressed in word form or a combination of both methods.

Definition 4.2. *(Value).*
A value $v \in \mathbb{R}$ *is a single number or a pair of numbers (lower and upper bound) representing a range* $v = \{[v_l, v_u] | v_l, v_u \in \mathbb{R}\}$. [2]

As mentioned previously not all numbers are considered a valid value of a quantity. For a number to be a part of a quantity, in addition to a value it should also have a unit.

- In the sentence "Apollo 7 took off", "7" has neither a unit nor explains an attribute.
- In the sentence "Apollo is 1.5 kilometers in diameter.", "1.5" has "kilometers" as a unit and is describing an attribute of "Apollo".

Table 4.1 illustrates different types of values typically present in the text. This list is by no means exhaustive, and new varieties might occur with a combination of two or more types.
Rational numbers cover a wide range of types and representations. Technically *word form*, *integers*,

[2] Open and closed intervals can be converted to one another by changing the lower and upper bound. For simplicity, we only consider closed intervals, where the lower and upper bound are included.

Table 4.1: Quantity value types with their examples.

Value type	Example	Standardized
Word form	hundred and thirty two	132.0
Mixed form	10k	10,000.0
Ratio	10 out of 20	0.5
Fraction	$\frac{1}{2}$	0.5
Integers	-22	-22.0
Natural numbers	12	12.0
Rational numbers	5.9	5.9
Exponential	$1.2E+4$	12,000.0
Range	$50-60$	[50.0, 60.0]
Math equation	$12+7-1$	18.0

natural, *fractions*, *ratios*, and *exponentials* are rational numbers with different textual representations. However, for the sake of completeness, we assign them to a separate category. The difference between integers and natural numbers is the sign, where integer values also include negative numbers. Word form values are the written-out version of a number, and mixture forms are a mix of word tokens and numerics values. Values come in different magnitudes, often denoted by prefixes or post-fixes, e.g., "200 million". The differences in magnitudes and scales are mainly shown by single characters in front or after a numeric in a mixture form representation. Fractions are sometimes written out in the text as ratios, e.g., "$\frac{1}{5}$" or "one-fifth". Ratios in some contexts can also imply a percentage (Roy et al., 2015), which makes the extraction of units difficult. The sentence "1 out of 10 patients developed post-corona symptoms" implies that "10%" of the patients developed post-corona symptoms.

Ranges denote a set of values between a lower and upper bound and are represented by the set of two real numbers. Lower and upper bounds can be represented by any of the value types described above, e.g., "2 to 4 million dollars" or "between 4.5 and 5.6". In some unconventional cases, upper bound comes first in the text, e.g., "between 4 and 2 pounds".

Math equations cover a broad range of complex mathematical equations, which can be further categorized. However, complex equations that also infer a unit rarely occur outside of physics and domain-specific text and are not the focus of this thesis. Hence, they are not considered by our CQE framework. If a unified representation for such equations is required, one could consider the outcome of solving the equation as the value.

Other forms of value types worth mentioning are calendar dates, e.g., "28.02.1991", and timestamps, e.g., "11:25". Date and time are often represented in many different surface forms and sometimes with a mixture of word forms and numerics. The different scales and specific date arithmetics make calendar time a challenging quantity in comparison to the rest. Our CQE framework does not consider calendar time. The reason for this decision is twofold. Firstly, the correct normalization of dates is often dependent on context that goes far beyond a single sentence and sometimes requires

additional information about the time of document creation. For example, "yesterday" is highly dependent on the date on which the text is written or read. Second, temporal expressions are ordinal values based on a given contract, varied across time zones, which makes them inherently different from other quantities considered by CQE. Extraction and normalization of temporal expression is best left to a system specialized for temporal tagging. There are various shared tasks and dedicated datasets created for temporal tagging and a vast area of research for their exploration in dynamic document collections (Hausner et al., 2020; Sousa et al., 2023b; UzZaman et al., 2013a; Verhagen et al., 2007, 2010). The traditional high-precision systems for temporal tagging were rule-based and rely on a set of hand-crafted features for the detection and normalization of temporal expressions (Chang and Manning, 2013; Strötgen and Gertz, 2015; Manning et al., 2014; Strötgen and Gertz, 2010). Recently, a number of fully supervised approaches have also aimed to solve the identification of temporal tags (Almasian et al., 2021, 2022a; Aumiller et al., 2022; Chen et al., 2019b; Lange et al., 2020; Laparra et al., 2018; Sousa et al., 2023a). However, the normalization still remains rule-based in most systems.
It is also important to make the distinction between concrete date-time mentions and an indication of durations. Durations like "6 seconds" or "5 hours" are not date-time mentions and do not require specific time scales. In CQE, we consider durations like the rest of the quantities.

4.2.2 Units

A unit is a noun or a noun phrase defining the atomic unit of measure for a quantity. Different units measure different physical properties and are used to communicate measurements among people (Rijgersberg et al., 2013). When referring to units often the global metric system comes to mind, which has a pre-defined set of units for weights and measurements, e.g., "2km" has the unit "kilometer". But the complexity of human language and how we reference quantities allows for almost any noun phrase to take the role of a unit, e.g., "2 apples", where apple is the unit. To account for all variations we divide the units into three categories:

1. *Scientific units (Si):* This is the most common type, containing systems of weights and measures. The agreement on the practical use of units has been an ongoing debate for years, due to the multitude of systems. However, the global standard is SI (Rijgersberg et al., 2013). Hence, scientific units are divided into SI and non-SI units. SI, short for International System of Units, comprises seven base units, which are:
    a) *meter* for distance
    b) *kilogram* for mass
    c) *second* for time
    d) *ampere* for electric current
    e) *kelvin* for temperature

f) *mole* for amount of substance

g) *candela* for the intensity of light.

The other SI units, called SI-derived units, are then defined algebraically in terms of these base units, by multiplication, division, and exponentiation of them (Thompson and Taylor, 2008; Rijgersberg et al., 2013). For example, "Newton" is a derived unit (in the SI) defined as "$kg\frac{m}{s^2}$". Even though SI is the recommended practical system, there are still several non-SI units that are commonly used, mainly due to historical and political reasons, as well as convenience. For example, in some countries distance is measured in "inches", "feet", "yards", and "miles", rather than "millimeters", "meters", and "kilometers". The representation of units in a text also differs, some units are represented by symbols and some by abbreviations. Moreover, other metric systems, such as the Centimetre–Gram–Second (Tittmann, 1892), MKS system of measurements (Nicholson, 1951) are still used, although no longer prominent.

2. *Currency (Cu):* A unit of money, such as the pound, dollar, euro, or even cryptocurrencies. They are present in text in word form, e.g., "euro", ISO 4217 currency code, e.g., "EUR", or currency sign, e.g., "€". All units of money can be converted to one another. The conversion is time-dependent, due to the volatility of exchange rates each day, thus, representing all the monetary values in a text with a single unit is challenging and outside of the scope of this study. In this work, we treat all the monetary units as they are and leave the task of time-dependent unit conversion to future work.

3. *Noun Phrase (Np):* Usually the noun or noun phrase after a numeric indicates the unit for that number. Although the scientific units are also nouns, in this category we only consider the nouns that are not in the pre-defined metric system. Quantities in this format are often used to count the multitude of the objects the noun is referring to. For example, in sentences "I ate 4 *apples*" and "1,500 *people* were attending the concert", *apples* and *people* are not pre-defined metrics but the objects that are being counted.

Scientific units and currencies can have many textual or symbolic surface forms, and their normalization is a daunting task, e.g., "km", "kilometer", "kilometre" or "euro", "EUR", "€". Occasionally the surface forms or symbols of a unit coincide with other units, resulting in ambiguity that can only be resolved by knowing the context, e.g., "She weighs 50 pounds" is a measure of weight ("pound-mass") and not a currency. Therefore, it is important for a quantity extractor to perform context-dependent disambiguation.

At times the unit can be inferred implicitly from the context, although not present in the text. For example, "She was born in 1991", is referring to the "year" of 1991. These mentions are more prominent for dates and times and are not considered in our quantity model, as explained above.

Definition 4.3. *(Unit).*
A unit u is a noun (single token) or a noun phrase (multi-token) defining the atomic unit of measure for a quantity, belonging to either the set of scientific units Si, set of currencies Cu, or a set of relevant noun phrases Np, $u \in U, U := Si \cup Np \cup Cu$.

A notable feature of scientific units is that, if correctly identified, they can be converted to standard base units, such as converting "1 mile" to "1609.344 meters". This allows values with different units referring to the same attribute to be compared through unit conversion. Unit conversion involves changing the value from one unit type to another using multiplicative conversion factors or addition and subtraction. The primary goal is to convert the value without altering the property being measured, typically converting to SI base units or a more common unit. However, these conversions are not always exact and may involve rounding.
Conversion is also possible for noun-based units but becomes more tedious, e.g., "Julia ate 4 apples and 3 bananas" indicates that Julia ate 7 fruits. Detecting such generalizations requires extensive knowledge of the taxonomy of all possible noun combinations and synonyms, which cannot be achieved without a lexical database such as WordNet (Miller, 1998). Even with WordNet synsets, the question of whether such conversions distort the meaning of the sentence or not persists. Moreover, some noun phrases contain numeric information, resulting in a composition of quantities and units. For example, "2 Italian couples", implies 4 people, or "Marry took 1h 30 minutes ride to get here", implies that Marry was 90 minutes on the way.
In our CQE framework, we focus solely on normalizing scientific units to their base SI units, without considering conversions among scientific or noun-based units. This decision is based on the fact that unit conversions are not always exact and can introduce unwanted errors in our experiments. Additionally, converting all units to a base unit would eliminate important granularities specific to certain domains. For instance, in a clinical setting, doses are often measured in small units like "millilitre" and "milligram". If unit conversion is required, several third-party libraries are available that can be applied on top of CQE output. [3] On the other hand, noun-based conversions require semantic reasoning and discovering hidden arithmetical laws among nouns, which is more related to linguistic and language comprehension and is outside of the scope of this thesis.
Scientific units can be combined with multiple suffixes and prefixes for naming very large or very small numbers, such as "micro" or "mega", representing a multiplication factor. The combination of prefix and unit is called a *multiple of a unit* (Rijgersberg et al., 2013), e.g., "megameter". Before the expansion of the prefix list in 1960, double prefixes were also allowed, such as "kilo mega". Although these units are rarely used, they are still present in some texts.

Sometimes units appear in combination as *Compound units*, which combine multiple units and measurements. Compound units are expressed as multiplication, division, or power of other units

[3] An example of a pip package is `https://pypi.org/project/unit-convert/.` (last accessed: 02.05.2024).

and can not be prefixed. Examples in a scientific context are "speed", which is defined using both "distance" and "time", and "density" as in "kilograms" divided by "meters cubed". Noun-based units can also be combined with scientific units, e.g., "rate of payment is £20/h" refers to "twenty pounds per hour". Surface forms of compound units are not always consistent and various separators are used for the combination. Common separators include:

- / , e.g., "km/hour"
- *per*, e.g., "kilometer per hour"
- *a*, e.g., "pound a gallon".

In rare cases, compound units may consist of more than two units, e.g., "mg/g/day".

Detecting a unit in a given text might seem like an easy task but it is by no means trivial. Even with scientific units where a specific list and taxonomy exist, the unfamiliarity of the writer with the rules or typos makes the task difficult. Some of the common problems are indicated as follows:

1. Symbols referring to a unit are not separated from the numerical quantity. In these cases, it is not obvious if the symbol is the suffix of the value of the prefix of the unit. For example, "5kg" is ambiguous and can be interpreted as "5 kilogram" or "5000 gram".

2. The slash ("/") in compound units is sometimes confused with backslash "\", or unnecessary spacing is added, e.g., "kilometre per hour" is written as "km\h" or "km \ h".

3. The superscripts "__^{2}" and "__^{3}", used for square and cubic are oftentimes not well-formatted in text and are written without superscription. As a result, they are confused with the value of the quantity, e.g., "kg/m3".

4. Not all writers respect the case sensitivity of units. There is a tradition that the first letter of an unprefixed unit symbol is capitalized if (and only if) the unit's name comes from a proper name. This is important because often the same letter represents different units: e.g., "t" stands for the "tonne" and "T" for the "tesla".

5. Inconsistent writing of confusing units. For example, "per cent", can be a "percent" or "for each cent" (as a unit of currency).

6. Occasionally, dashes and spaces are inserted between units and values, e.g., "12-cent-a-share" or "200sq m". There are other unnecessary symbols that are often added between units and values, e.g., "100-million", "100_million", or using multiple numbers with the same suffix, e.g., "100, 200, and 300 million".

Table 4.2: Types of changes with examples and keywords.

Change	Example	keywords	Representation
equal	2	exactly, just, equals, totalling, =	=
approximate	around 2	roughly, nearly, approximately, about, round about, nearly, circa, barely average, averaging, close, almost, estimate, some, even, ∼ , =∼, ≈	∼
greater than	more than 2	greater, great, above, larger, large, higher, high, over, above, exceeding exceed,least, well over, >, >=	>
less than	less than 2	smaller, below, up to, under, no more than, beneath lower, low, beneath, fewer, few, less, <, <=	<
increasing	increased to 2	add, advance, ascend, climb, gain, grow, increase, jump rally, raise, rise under, surge, up, upward	*up*
decreasing	dropped to 2	collapse, decline, decrease, descend, dip, down, downward, shrink, spil, fall drop, flop, lose, plummet, plunge, sag, skid, slide, slump, stumble, tumble	*down*

7. Using unconventional abbreviations is another common issue, like “bn” for “billion” or “degf” for “fahrenheit”. This is most prominent in medical texts, where the abbreviation of one doctor might differ from the next one. For example, in a single corpus “y”, “yr”, and “yo” are all used as abbreviations for “years of age”.

8. Traditional unit abbreviations are still used and only add to the confusion. The letter “p” is often used as an abbreviation for “per”, “sq” or “s” as an abbreviation for "square", and “cu” or “c” as an abbreviation for “cubic” (Thompson and Taylor, 2008).

9. An official symbol is hard to find, therefore the writer replaces it with a made-up one, or due to format conversions between documents, the true symbol is lost, e.g., the symbol “£” is converted to “PS” when HTML4 files with a character encoding of ISO-8859-1 are converted to UTF-8 character encoding.

10. The abbreviation of a unit is the same as the metric symbol of another. For example, “A” is sometimes used in English for the “acre” instead of the “ampere”.

11. Multiple spelling of the same unit exists for American and British English, e.g., in American English one writes “meter” and “liter” while the British variant is “metre” and “litre”.

4.2.3 Changes of Values

Changes are modifiers of values, describing how a value is changing or imposing a bound on a value. For example “above 35$” is describing a bound with values more than a certain threshold, whereas “the temperature dropped 10 degrees“ shows a changing trend. Table 4.2 contains all changes considered in this work with examples and representations. Changes are often identified using specific keywords before the value, e.g., “more than 5 percent”. The keyword list in Table 4.2 is not comprehensive and any keyword that indicates a modification to the quantity value can be part of this

list. It is worth noting that our definition of change does not cover negation, e.g., "the shirt did not cost 20 euros", as we do not look into the semantics of the entire sentence to derive the changes. We introduce six categories for change, where the first four are mainly describing the bounds for a quantity and the last two are accounting for trends:

1. *Equal (=)*: These are changes that indicate exact values, e.g., "the laptop costs 1000 euros".
2. *Approximate (∼)*: These are values that are not exact and are approximate, e.g., "the laptop costs around 1000 euros".
3. *Greater than (>)*: These are bound-based changes that indicate a lower bound for a range of values, e.g., "the laptop costs more than 10 euros".
4. *Less than (<)*: These are bound-based changes that indicate an upper bound for a range of values, e.g., "the laptop costs less than 1000 euros".
5. up: These are changes that indicate an increasing trend, where the values are rising, e.g., "DAX increased by 2%", in this case, the DAX value has an increase of 2 percent.
6. $down$: These are changes that indicate a decreasing trend, where the values are falling, e.g., "DAX fell 2%", where DAX is losing its value.

Definition 4.4. *(Change).*
A change describes the alteration in a value of a quantity and describes exact, approximation, more than, less than conditions as well as upward and downward trends, with $ch \in \{=, \sim, >, <, up, down\}$.

Greater than and less than bounds inherently describe a range of values, e.g., "the laptop costs less than 1000 euros" shows the range of values between zero and 1000 that are valid. In such cases, we consider a change independently of the value and store the contextualized quantity with $v = 1000$ and $ch = '<'$. The range of value can later be constructed using contextualized information.

4.2.4 Concepts

Concepts are objects or events that a quantity is referring to. A quantity mentioned in a text is either measuring a property of an object, e.g., "height of the Eiffel Tower", in which case the phenomenon and the property are the concepts, or an event involving a quantity, e.g., "Google hired 100 people", in which case the actor is what the quantity is referring to. The detection of concepts is a tedious task and requires concentration and language understanding even for a human to correctly identify the related token in a given text. We make the following observations regarding concept detection:

- If the quantity is directly measuring a property of an object, then the concept is *direct*. In the phrase "German DAX fell 2%" the quantity is measuring the worth of the "German DAX".

- *Indirect* concepts may be also present in a text, where a quantity is referring to an action made by an entity. In the sentence "The BMW Group is investing a total of $200 million", the investment is being made by the "BMW Group".

- Sometimes concepts are distributed in different parts of a sentence, e.g., in the sentence "The iPhone 11 has 64GB of storage." both "iPhone 11" and the "storage" are the concepts.

- Occasionally, concepts are a mixture of direct and indirect references. For example, in the sentence "The radiologist recommended limiting the patients' exposure to no more than 0.5 R per day" the quantity refers to both the "exposure of patients" directly and the "radiologist who prescribes it" indirectly.

- There are cases of *shared concepts*, where a single concept can refer to multiple quantities in text. In the sentence "Screen sizes remain 5.4 inches, 6.1 inches and 6.7 inches for the mini, standard and Pro Max models, respectively" all three quantities refer to "screen size".

- A concept may or may not be present in the text. The sentence "2,000 people went to the concert." does not have a concept, "people" is the noun unit of the quantity, and there is no external entity that "2,000 people" is referring to. In other words, the object being talked about (subject of the sentence) is the quantity itself. Nevertheless, if we change the sentence slightly to " The capacity of the concert hall is 2,000 people", now "2,000 people" is referring to the concept "concert hall".

- For correct identification of unknown pronouns as concepts, co-reference resolution is required, e.g., "She has completed sprint triathlons including a 750 meter swim", where "she" is an unknown pronoun that can only be deduced from the previous context. In this study, we assume that co-reference resolution has been done prior to quantity extraction and such pronouns are already resolved.

Definition 4.5. *(Concept).*
Given a sentence s containing a quantity q, the function $Rel(s, q)$ maps the concept related to q, as a set of tokens, $cn = \{t | t \in s \wedge Rel(s, q) = t\}$. If no concept is present, $Rel(s, q) = \{\}$.

4.3 Comprehensive Quantity Extraction (CQE)

Different elements of a contextual quantity appear in various formats in text, which is highly dependent on the topic of the text and the writing style of the author. Nonetheless, with the knowledge of different formats, quantities often follow a set of recurring patterns that help distinguish them with a set of rules.

The goal of this section is to a design the $Extract$ function in Definition 4.1. There are many ways to design the $Extract$ function. However, due to the defined formatting of quantities most previous works rely mainly on regular expressions (Alonso and Sellam, 2018; Banerjee et al., 2009; Maiya et al., 2015). We follow a similar strategy but instead of relying on regular expressions, we take advantage of linguistic properties and dependency parsing. We justify the use of a rule-based system against a supervised learning solution as follows:

- One of the major motivations for using a rule-based approach is the lack of data for comprehensive quantity extraction. There are no datasets available for training or testing quantity extraction systems. A few systems that rely on a supervised approach for unit and value extraction do not provide their test or train datasets, and in some cases, even the statistics of the data are unknown (Foppiano et al., 2019; Roy et al., 2015).

- Quantities often follow a specific linguistic pattern in the text, which is easily captured by a set of rules. Moreover, a rule-based system provides transparency, high control over the output of the model, and easy error analysis.

- In Chapter 5, contextualized quantities are used to create quantity-aware retrieval systems. For this purpose, we require a system with high precision that does not need fine-tuning to a specific data distribution, which is often the case for supervised learning approaches. In case of a rule-based system, any domain specific writing style can be captured in a set of new rules and added to the system.

- At the time of conducting these experiments, large language models such as GPT-3 (Brown et al., 2020) with few shot learning capabilities still make numerous mistakes on quantity extraction; More details on this are provided in Section 4.4.

A comprehensive quantity extractor takes a sentence as input and produces a tuple of value, unit, change, and concept for each quantity in the sentence. For example, the sentence "In Europe, German DAX fell 0.4 pc, while the CAC40 in France gained 0.1" results in two quantities:

- $\langle v = 0.4,\ u = percentage, ch = down,\ cn = \{German, DAX\}\rangle$

- $\langle v = 0.1,\ u = percentage, ch = up,\ cn = \{CAC40, France\}\rangle$.

There are two reasons why we decided to create our own extractor and not rely on existing packages. First, most extractors in the literature are not open-source, and the few open-source libraries that exist lack any quality assurance. Due to the lack of a benchmark dataset for this task and evaluation only through downstream performance, choosing the most effective extractor becomes merely an exploratory task. This brings us to the second point. After testing the available libraries and studying their shortcomings, we concluded that none can satisfy the conditions that we need for our subsequent downstream application in Chapter 5. More details around this topic are provided

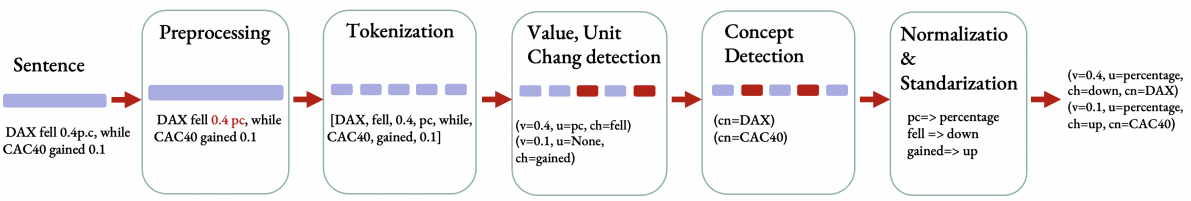


Figure 4.1: Five stages of comprehensive quantity extraction with the example sentence of "DAX fell 0.4p.c., while CAC40 gained 0.1".

in Section 4.4.1, where different systems are compared. The mentioned reasons are also our motivation for proposing the first-ever quantity extraction benchmark dataset, named NewsQuant.

Our proposed system (CQE) performs the extraction in five stages, as described next. A visualization of the entire pipeline is shown in Figure 4.1, and the steps are detailed in the following subsections.

4.3.1 Pre-processing

The input of CQE is a sentence. We assume sentence splitting has been performed prior to quantity extraction. Pre-processing and text cleaning are done prior to dependency parsing and POS-tagging, alleviating some of the style issues mentioned in previous sections. The main pre-processing steps are as follows:

- Removal of unnecessary punctuation and spacing. Certain punctuation are common but misplaced in units. Examples:
  - "400 million-year-old" → "400 million year-old".
  - "m.p.h" with removal of dots → "mph".
  - "200sq m" has an unnecessary spacing between the unit tokens → "200 sqm"
- Adding helper tokens. Some tokens are transformed to a standard format to fit the rules for correct detection of values and units. Examples:
  - "sub-500 sqm", where *sub* is an unusual word for *under* → "under 500 sqm"
  - "US71.37", where the unit is attached to the value, a space is required → "US 71.37"
  - "+0.2%", where the plus sign indicates a upward trend → "up 0.2%"
- Tagging date and time entities to be ignored. Since date and time values do not fit the definition of our quantities, they need to be detected and ignored. Examples:
  - day of the year mentions, such as "12 Sep"
  - year mentions, such as "She was born in 1991"

- Removal of numeric values that do not fit the definition of a quantity, such as phone numbers and zip codes. These values follow a general pattern and are easily recognized with regular expressions. Examples:
  - "205 Mathemtikon, Im Neuenheimer Feld, 69120"
  - "+49 (0) 6221 / 54 - 14353"

4.3.2 Tokenization

We perform a custom task-specific word tokenization. As mentioned in Section 2.2.4, available word tokenizers are focused on general domain text, and since numbers take up a small portion of most corpora, little work has been done to customize tokenizers for quantities. Our tokenizer is aware of separator patterns in values and units and avoids between-word splitting to keep the unit and value as single tokens. For example, in the sentence "A beetle goes from 0 to 80 km/h in 8 seconds.", a normal tokenizer would split "km/h" → "(km, /, h)" but we will keep the entire unit intact. Another example is numerical tokens containing punctuation, e.g., "$2.33E$-3" or "110,000", where naive tokenization splits the values.

4.3.3 Value, Unit, and Change Detection

The tokenized text is matched against a set of rules based on a dependency parsing tree and POS-tags. A set of 61 rules was created based on patterns observed in financial data, a small set of scientific documents and medical doctoral letters and also by considering previous work by Maiya et al. (2015) and Huang et al. (2017). The rules are designed to detect tokens in a sentence associated with value/unit pairs and change.

Value/unit pairs are often (1) numbers and nouns, (2) numbers and symbols, or (3) numbers and adjectives that co-occur in various sentence structures:

1. Numbers and nouns, e.g., "400 people", "55 square meter" and "30 Mbps"
2. Numbers and symbols, e.g., "1M €" or "$100"
3. Number and adjectives, e.g., "2.7 million contract" or "600 people worldwide"

For ranges, the extraction of values becomes more complex, as lower and upper bounds need to be identified using relational keywords such as "from... to" or "between", e.g., "between 5% and 6%". Occasionally the unit or part of the unit between the lower and upper bound is implicitly shared. For example, in the phrase "between 15kHz and 17", not only the unit but the scaling prefix is shared between the lower and upper bound, or in the phrase "between $62 and $68 per share" part of the unit is mentioned explicitly (dollar sign) but *per share* is implied.

Table 4.3: Extraction rules for value/unit pairs and changes and their structure.

Extraction Target	Number of rules	Example
change as noun, value as number, noun as unit	5	lower than 8.90 ksi
percentage of a noun	2	60% of workers
values with symbols	2	$62
value as number, noun as unit	4	2.4 Ghz
value without unit	1	thirteen
value as number, noun phrase as compound unit	8	14 first-team players
value as number, adjective as unit	3	90 day old
value as number, changes as dependent of the value	3	only 799
negative values	2	−1.7%
value as number, changes as verbs	2	fell by 25
values as number that share the same unit	1	$62 and $68 per share
numbers that are tagged as nouns	2	tens of thousands
value with postfix	2	7.2 billion
ranges that have “to” in between	5	$920 to $1730 a week
ranges that have “-” in between	3	5-10
range that have “between” or “from”	8	from 5.7% to 3.4%
implicit ranges	2	tens of thousands of students
fractions	2	three out of five
spelled out values	3	five hundred
one of a noun	1	one of the computers
Σ rules	61	

Changes are often adjectives, e.g., “ about 600” or verbs, e.g., “ climbed 0.1%” that have a direct relation to a number and modify its value. Sometimes symbols before a number are also an indication of a change, e.g., “∼ 10” describes an approximation. The list of changes considered in CQE is mentioned in the previous section in Table 4.2, along with the list of keywords. If one of those keywords or symbols is occurring with dependency relation to a number the change is allocated to the quantity.

For an exhaustive list of all 61 rules, we refer the curious reader to our open-source library, where a comprehensive list and examples are available. [4] As an overview, the statistics of number of rules, grouped by the elements they extract are given in Table 4.3. There is no execution order for the rules, as these are detection rule set and make no changes to the input. All the rules are applied simultaneously to extract possible candidates that are further processed in later stages. We would like to point out that this set is not complete and based on domain of the text, and the language, new rules need to be added to the list. Moreover, inclusion of certain rules is subjective, e.g., in the sentence “one of the computers”, it is debatable if “one computer” should be extracted or not.

[4] https://github.com/satya77/CQE (last accessed: 02.05.2024).

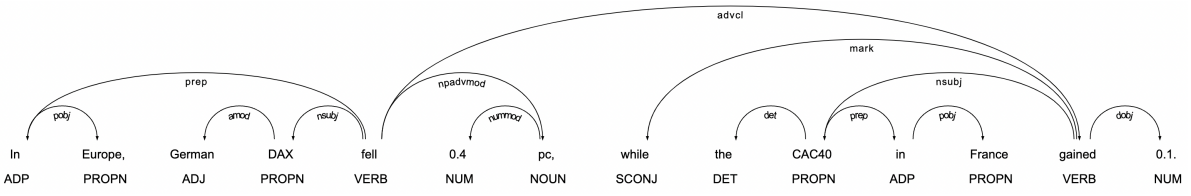


Figure 4.2: Dependency parsing tree of "In Europe, German DAX fell 0.4 pc, while the CAC40 in France gained 0.1.".

To understand how an extraction is performed we look at the detection of value, unit, and change in the sentence "In Europe, German DAX fell 0.4 pc, while the CAC40 in France gained 0.1.". The dependency parsing tree of this example is shown in Figure 4.2. Note that in this stage only the surface forms (spans in the text associated with each attribute) are detected and normalization to a standardized format is performed in later stages.

- *Value and unit of the first quantity:* The NOUN_NUM [5] rule detects the surface form of the first value/unit pair (0.4, pc). According to this rule, the numeric value should have a POS tag of NUM and be the immediate syntactic dependent of the unit token, which should be a noun or proper noun. When such a pair is found, the respective tokens are marked as value/unit pairs.
- *Value and unit of the second quantity:* The LONELY_NUM rule detects the value/unit pair for the second quantity, namely (0.1, {}). If all other rules fail to identify a value/unit pair, indicating the absence of an explicit unit for the quantity, this rule detects the number with a POS-tag of NUM. In such cases, only a single value is detected without an accompanying unit, and the unit is inferred in subsequent steps.
- *Changes for both quantities:* The QUANTMOD_DIRECT_NUM rule detects the change, by looking at the verb or adjective directly before tokens with POS-tag of NUM. Here, "fell" is a trigger word for a downward trend, and "gained" is a trigger word for an upward trend.

We thus have two extracted triplets with value, unit, and change.

- $\langle v = 0.4,\ u = pc,\ ch = fell \rangle$
- $\langle v = 0.1,\ u = \{\},\ ch = gained \rangle$,

If no unit is detected for a quantity, its context is checked for the possibility of *shared units*. For the quantity $\langle v = 0.1,\ u = \{\},\ ch = gained \rangle$,"percentage" is the derived unit, although not mentioned in the text.
Shared units often occur in similarly structured sub-clauses or after connector words such as "and", "while", or "whereas". Authors tend to omit writing out the second unit as it is implied by the

[5] The names of the rules are preserved from the repository rule set for easier matching.

sentence structure. In our method, the similarity between two sub-clauses is computed using the *Levenshtein ratio* between the structure of clauses. The structure is represented by POS-tags, e.g., "German DAX fell 0.4 pc" → "`JJ NNP VBD CD NN`" and "the CAC40 in France gained 0.1" →"`DT NNP IN NNP VBD CD`". The similarity between two clauses increases as their POS-tags follow the same order. By representing the structure as a sequence of POS-tags, the Levenshtein ratio can be used to measure their similarity. This ratio ranges from 0 to 100, with higher values indicating greater similarity. We infer a shared unit between two sub-clauses if their similarity is 100 or if connector words are present and the ratio exceeds 60. As a result, the unitless quantity is assigned the unit of the other sub-clause, e.g., in the mentioned example "{}" becomes "pc".

As a final step in this stage, the candidate values are filtered by logical rules to avoid false detection of *non-quantities*, e.g., in "S&P 500", 500 is not a quantity but part of the stock index name.

Further extraction examples

In this section, we provide two additional examples of value, unit, and change detection and describe the logic behind a few other rules. If the reader finds the single example from the previous section sufficient, this section can be skipped.

"The Meged field has produced in the past about 1 million barrels of oil, but its last well was capped due to technical problems that have not been resolved."

- *Value detection:* The NUM_NUM rule detects the compound number of 1 $million$, where 1, a number, is the child of $million$, a noun, in the dependency tree.
- *Unit detection:* The NOUN_NUM_ADP_RIGHT_NOUN rule finds a noun or a proper noun with a number as a child in the dependency tree. If there are prepositions connected to the candidate noun, they are also considered as part of the unit. In this case, the candidates are: $\{million,\ barrels,\ of,\ oil\}$.
- *Change detection:* The QUANTMOD_DIRECT_NUM rule detects the relation between the adjective "about" to the value 1, which is identified as the change.

From the combination of all rules, the candidate tokens $\{about\}$ for change, $\{1,\ million\}$ for value and $\{barrels,\ of,\ oil\}$ for unit are extracted.

"They pay a $3500 a month mortgage and two kids in private school."

- *Value detection for the first quantity:* NUM_SYMBOL matches a symbol followed by a number. In this case, $3500 is detected.

- *Unit detection for the first quantity:* NOUN_NUM_QUANT finds a number with a noun or an adverb as its head in the dependency tree. Here, we have $\{mortgage,\ 3500,\ \$,\ month\}$, where 3500 is the number and $\{mortgage, \$,\ month\}$ are the nouns.
- *Unit detection for the first quantity:* UNIT_FRAC_2 finds compound units with "per", "a" or "an" in between. In case of our example, $\{\$,\ month,\ a\}$.
- *Value and unit detection for the second quantity:* NOUN_NUM detects a noun that has a number as a child. In case of our example, "$\{kids,\ two\}$".

If no change keyword is found in a sentence, the default change is "equal". The mentioned rules contribute to the extraction of two candidate contextual quantities.

1. $\{\$,\ 3500,\ a, month,\ mortgage\}$
2. $\{two,\ kids\}$

Note that if no tokens indicating a change are detected in this case, the system defaults to "equal" in the normalization stage.

4.3.4 Concept Detection

Concepts are detected using a separate rule-set, containing five rules. Unlike the rules for value, unit and change detection, these rules are ordered by priority. If one rule manages to detect a concept, later rules are ignored. The rules are as followed:

1. Keywords, such as "for", "of", "at", or "by" before or after a numeric value point to a potential concept. In "with carbon levels *at* 1200 parts per million" the concept $cn = \{carbon,\ levels\}$ is hinted at by the keyword "at". The nouns before and after such keywords are potential concepts. These keywords usually indicate a relationship between a property and a quantity.
2. When a number appears as a leaf node in a dependency parsing tree, the entire subtree is examined to identify the closest verb to the number. If no verb is found, the verb connected to the ROOT node is chosen instead. This verb serves as a seed to determine the correct concept, with the nominal subject of the verb being considered as a potential concept. In the sentence "In Europe, German DAX fell 0.4 pc, while the CAC40 in France gained 0.1.", both "German DAX" and "CAC40 in France" are the nominal subjects of the closest verbs to the number tokens in the text. In such cases, an entity or object is the nominal subject of the sentence, which has a quantitative property and the sentence indicates this relationship.
3. Sometimes values occur in a *relative clause* that modifies the nominal. In the sentence, "maximum investment per person, which is 50000", the value is mentioned in the relative clause

to the concept $cn = \{maximum,\ investment,\ per,\ person\}$. In such a case, the noun phrase before the relative clause is the concept, since the relative clause describes it. These concepts occur when a relative clause to an object or entity describes one of its quantitative properties.

4. If the numerical value in a sentence is not associated with the nominal of the sentence, then it is mostly likely related to the object of the sentence. Therefore, the direct object of the verb is also a candidate concept, e.g., "She gave me a raise of $1k", where "raise" is the direct object of the verb. In this case, the value does not describe the person or object performing the action; instead, it pertains to the object upon which the action is performed. This structure is usually present in a passive sentence, which is less frequent and is, therefore, lower in the priority list of extraction rules.

5. Finally, if a concept is not found in the previous steps, and there is a single noun in the sentence, the noun is tagged as the concept, e.g., "a beetle that can go from 0 to 80 km/h in about 8 seconds." the concept is $cn = \{beetle\}$. Although this step can introduce unwanted errors in the concept extraction, we decided to keep it for a more recall-oriented approach. In our experiments, keeping such nouns increases the recall significantly.

From the list of candidate tokens for concepts stopwords are removed, e.g., "CAC40 in France" results in $cn = \{CAC40,\ France\}$.

4.3.5 Normalization and Standardization

The final stage is the normalization of units and changes and standardization of values.

Normalization of Units: The *units dictionary* is a set of 557 units, their surface forms and symbols gathered from (1) the dictionary of units in Quantulum3 library, (2) a dictionary provided by Unified Code for Units of Measure (Lefrançois and Zimmermann, 2018), and (3) a list of units from Wikipedia. [6] Other units were added as encountered in specific medical, financial, and sports domains. For the complete list, we refer the reader to the open-source repository of our project. Although this list is not comprehensive, if a user encounters a new unit, it can easily be extended by adding a new entry to the dictionary. Without further changes to other steps, the normalized form is then extracted. An example of an entry in this dictionary for "euro" is:

```
{"euro":
"entity": "currency",
"surfaces": ["Euro", "Euros", "euro", "euros"],
"symbols": ["EUR", "eur",€]}
```

[6] List of units in Wikipedia: https://en.wikipedia.org/wiki/Template:Convert/list_of_units (last accessed: 02.05.2024).

Each entry has three attributes.

1. *Surfaces* refers to the possible textual surface forms of the unit.
2. *Symbols* contains a set of symbols assigned to the unit.
3. The *entity* attribute organizes the units into 53 classes, e.g., "ratio", "angle", "length", "mass", and "volume". These classes categorize the units into a hierarchy and define the type of property they measure. This attribute can be used to group quantities of the same type, an option we explore in Chapter 5 for automatic data generation.

The candidate tokens detected (from the previous step) for a unit are normalized by matching against the different surface forms and symbols in the dictionary. The normalized form is the *key* of the dictionary and is added to the output, e.g., "euro" in the example above. Another example is in the phrase "12cm", the surface form "cm" results in the normalized unit of "centimetre". The normalization makes the comparison of units with various representations easier. Note that conversions between metric units are not supported. For example, "centimetre" is kept as the final representation and not converted to "metre".

If a detected surface form is shared across multiple units, the unit is *ambiguous* and requires further normalization based on the context. For example, "pound" is a surface form for both the currency of Great Britain and a unit of mass, or "B" can refer to both "byte" or "bel". The distinction between the different units is only possible based on the context and recognition of the type of object the quantity is referring to. Since language models are great at capturing contextual information, for this purpose, we train a BERT-based classifier (Devlin et al., 2019b). Basically, a BERT model is used to create an embedding of the input sentence and the embedding of the `[CLS]` token is fed into a linear layer for binary classification between the two ambiguous units. There are 18 ambiguous surface forms in our unit dictionary, and for each, a separate classifier is trained that allows us to distinguish among units based on the context. If an ambiguous surface form is detected by the system, the relevant classifier is used to find the correct normalized unit, e.g., one classifier separates "byte" and "bel" and one separates "pound-currency" and "pound-mass".

Compound units, where a unit is a combination of two or more independent units, are also detected and normalized independently. For example, "kV/cm" results in "kilovolt per centimetre", where "kV" and "cm" are normalized based on separate dictionary entries. Recognizing "kV" and "cm" as separate tokens relies on our custom tokenization described in the previous section.
If the candidate span of tokens does not match a unit in the dictionary, it is tagged as a *noun-based unit* and lemmatized, e.g., "10 students" gives $u = student$. The lemmatized form is considered as the normalized unit for a noun-based unit. In some cases, the adjective before a noun is also part of the unit, e.g., "two residential suites" results in $u = residential\ suite$. Noun-based units include

non-conventional units and are mainly ignored by the previous quantity extraction systems.

Standardization of Values: The *value dictionary* contains the necessary information to standardize values to real numbers. More specifically, it contains the following information:

- Prefixes and suffixes of scales that appear before or after a value, e.g., "B: billion" or "n: nano".
- Spelled out numbers in the textual format up to a hundred, e.g., "forty-two: 42".
- Fractions in textual format, e.g., "half: 1/2".
- Scientific exponents in textual format and encodings, e.g., "10^2: 100".

Moreover, scientific notations with exponent and mantissa are also recognized with regular expressions and converted to decimal values, e.g.,"2.3E2" is converted to $v = 23$".

Normalization of Changes: Various trigger words or symbols for bounds and trends are managed in the *changes dictionary* (refer to Table 4.2). The candidate tokens for change are mapped to one of the allowed categories based on the trigger words. In the sentence "Apple company earnings exceed 1 billion euros for the second quarter.", the trigger word "exceeds" points to the bound-based condition of more than, ">".

4.4 Evaluation

As already mentioned in the introduction of this chapter, one of the main challenges in finding a suitable quantity extractor is the lack of a benchmark dataset and standalone evaluation framework for such a system. To this end, we create the first comprehensive evaluation dataset from financial news articles and perform a detailed comparison with other quantity extractors that have a functioning code base. CQE is compared against *Illinois Quantifier* (Roy et al., 2015) (abbreviated to *IllQ*), *Quantulum3 (Q3)*[7], *Recognizers-Text (R-Txt)* (Huang et al., 2017), *Gorbid-quantities (Grbd)* (Foppiano et al., 2019), and GPT-3 with few-shot learning (Brown et al., 2020). From here on, the abbreviations are used to refer to the respective system. Before describing our novel benchmark dataset, we first compare the models based on their functionality. For a quantitative comparison, the models are compared on precision, recall, and F1-score for quantity extraction. The unit disambiguation module is evaluated separately, on a custom-made dataset against the only other extractor capable of unit disambiguation (Q3). The CQE package can be accessed under the url: `https://github.com/satya77/CQE` and our evaluation scripts and the NewsQuant dataset are available at: `https://github.com/satya77/CQE_Evaluation`.

[7] Quantulum3 package: `https://github.com/nielstron/quantulum3` (last accessed: 02.05.2024)

Table 4.4: Comparison of functionality for five extractors.

Feature	Example	CQE	IllQ	R-Txt	Q3	Grbd
Value	5k euros (5k)	✓	✓	✓	✓	✓
Standardization	5k euros (5000)	✓	✓	✓	✓	✗
Negative Values	−5 C (−5)	✓	✗	✓	✓	✓
Fractions	1/3 of the population (0.33)	✓	✗	✓	✓	✓
Range	40-60 km/h (40-60)	✓	✗	✗	✓	✓
Non-quantities	iPhone 11 (-)	✓	✗	✗	✗	✓
Scientific Notation	1.9×10^2 (190)	✓	✗	✗	✓	✗
Unit	1mm (mm)	✓	✓	✓	✓	✓
Unit normalization	1mm (millimetre)	✓	✗	✓	✓	✗
Unit disambiguation	10 pound (sterling or mass?)	✓	✗	✗	✓	✗
Noun Units	200 people (people)	✓	✓	✗	✗	✓
Shared Units	about 8 or $9 (both dollar)	✓	✗	✗	✓	✓
Change	more than 100 (>)	✓	✓	✗	✗	✗
Trends	DAX fell 2% (down)	✓	✗	✗	✗	✗
Concept	AAPL rose 2% (AAPL)	✓	✗	✗	✗	✗

4.4.1 Comparison of Functionality

Since quantity extractors are developed as a medium to support a certain downstream application, often on a specific domain of data, these models are also limited in their capability. A trend visible in all systems is a focus on narrow domains or limited types of quantities. With CQE, the aim is to overcome such limitations and have a system that combines the strength of available extractors. Table 4.4 compares the functionality of available systems in terms of different types of values, units and changes, as well as normalization techniques. These functionalities were determined by studying the systems mentioned in the table and also quantities in financial documents and are not comprehensive. Some of the key observations are as follows:

- IllQ is the only baseline capable of detecting changes in values, but it is limited and does not account for upward or downward trends. It performs partial normalization, focusing only on currencies and neglecting scientific units. Additionally, IllQ cannot detect fractional values and ranges.

- After our approach (CQE), Q3 has the most functionality and is the only model that correctly detects ranges and shared units and performs unit disambiguation. Yet, Q3 disregards noun-based units and only focuses on the units that are part of its internal dictionary. Although Q3 is capable of detecting a wide range of value types, it still makes incorrect detections of non-quantitative values, confusing entity names and postal codes with quantities.

- R-Txt has dedicated models for certain quantity types, which gives rise to good performance for those specific types. More specifically, R-Txt has models for currencies, dimension, tem-

perature, and age, but fails to detect other types of quantities in the text. Moreover, it cannot distinguish automatically between the types and the model has to be chosen beforehand, i.e., to detect currencies the currency model needs to be activated, and for dimensions, the dimension model. In order to detect all possible quantities, all four models need to be applied to a piece of text, which not only adds computational overhead but also creates ambiguity in cases where multiple models detect the same quantity but with different types.

- Grbd's major shortcoming is the lack of value standardization, where fractions such as "1/3" and scaled values like "2 billion" are not standardized. The system is limited to scientific units, and unit normalization works differently than other systems. Their definition of unit normalization is *conversion* of scientific units to the base SI unit, and scaling of the values accordingly. Furthermore, Grbd fails to detect scientific notations and cannot disambiguate between units with similar surface forms.

CQE aims to address the shortcomings of the mentioned models and is the only model capable of identifying concepts and trends.
GPT-3 has a lot of variability in the output and does not provide concrete and stable functionality like the models discussed in this section. Therefore, it is not considered in this comparison.

4.4.2 Datasets

For evaluation purposes and training of the disambiguation classifiers in our research, two datasets were created. In the following subsections, the data collection and annotation strategies are discussed along with the statistics of each dataset.

4.4.2.1 NewsQuant Dataset

For a quantitative comparison, we introduce a new evaluation resource called NewsQuant, consisting of 590 sentences from news articles in the domains of economics, sports, technology, cars, science, and companies. To the best of our knowledge, this is the first comprehensive evaluation set introduced for quantity extraction. The sentences were chosen to test the capabilities mentioned in Table 4.4. Each sentence is tagged with one or more quantities containing a value, unit change, and concept. The dataset is annotated by the author of the thesis and a student contributing to the creation of CQE. Inter-annotator agreements are computed separately for value, unit, change, and concept on a subset of 20 samples. The Cohen Kappa coefficient (Cohen, 1960) of value, unit, change identification are 1.0, 0.92, and 0.85, respectively. Value detection is a simpler task for humans, and annotators have a perfect agreement, whereas unit and change detection requires a deeper understanding of unit surface forms and semantics of a sentence.
A concept is a span of tokens in the text and does not have a standardized representation, therefore the Cohen Kappa coefficient cannot be used. Instead, we report Krippendorff's alpha (Krippendorff, 2018), with the value of 0.79. The annotation guidelines are designed based on the definitions

Table 4.5: Statistics of the number of sentences, quantities, and sentences with and without quantities in the NewsQuant and R-Txt datasets.

Dataset	#sent	#quantity	#sent with quantity	#sent w/o quantity
NewsQuant	590	904	475	115
R-Txt-currencies	180	255	178	2
R-Txt-dimension	93	121	77	14
R-Txt-temperature	36	34	34	2
R-Txt-age	19	22	18	1

of value, unit, change, and concept described in previous sections. Additionally, the annotators were provided with the dictionary of units and values for purposes of normalization. As a summary of agreement over the entire sample set, we compute the percentage of times both annotators agreed on all attributes of a quantity. In this case, we reach the total agreement for 62% of the annotations.

For additional evaluation resources, the R-Txt repository contains a set of annotated quantities for age, dimension, temperature, and currency. [8] These datasets are also added to the evaluation set. However, they contain only unit/value pairs and no information about change or concepts. Moreover, the original dataset in its raw format is unsuitable for evaluation, as it only contains tags for certain quantity types and would ignore other types, giving the R-Txt model an advantage. For example, in the R-Txt-currencies, only the currencies were annotated and other quantities were ignored. For that purpose, all these datasets were revised, and extra annotations for all other types of quantities were added for a fair comparison. For example, in the sentence "I want to earn $10000 in 3 years", where only "$10000" was annotated, "3 years" is added.
Table 4.5 shows the statistics of the number of sentences and quantities for each dataset. The NewsQuant dataset is the largest dataset for this task containing over 900 quantities of various types. NewsQuant also includes 74 negative examples with non-quantity numeric values. Three examples from the data are shown below:


```
{"text": "Unregulated car finance falls outside the Consumer Credit Act 1974 leaving
those who want to end their car finance agreements early part-exchange the vehicle
or pay off the total value facing large penalties.",
"quantities": []},

{"text": "The Dow Jones Industrial Average lost 190 points, or 0.6%. The S&P 500 fell
0.9%, while the Nasdaq Composite slid 1.1%.",
"quantities": [{"change": "down",
                "value": "190.0",
                "unit": "points",
                "normalized_unit": "point",
                "referred_concepts": "Dow Jones Industrial Average"},

               {"change": "down",
```


[8] Recognizers-Text datasets: https://github.com/microsoft/Recognizers-Text/tree/master/Specs/NumberWithUnit/English (last accessed: 02.05.2024)

```
                "value": "0.6",
                "unit": "%",
                "normalized_unit": "percentage",
                "referred_concepts": "Dow Jones Industrial Average"},

               {"change": "down",
                "value": "0.9",
                "unit": "%",
                "normalized_unit": "percentage",
                "referred_concepts": "S&P 500"},

               {"change": "down",
                "value": "1.1",
                "unit": "%",
                "normalized_unit": "percentage",
                "referred_concepts": "Nasdaq Composite"}]},

{"text": "Apple One's Individual plan is $14.95 a month and comes with 50GB storage.",
"quantities": [{"change": "=",
                "value": "14.95",
                "unit": "$ a month",
                "normalized_unit": "dollar per month",
                "referred_concepts": "Apple One Individual plan"},

               {"change": "=",
                "value": "50.0",
                "unit": "GB",
                "normalized_unit": "gigabyte",
                "referred_concepts": "storage, Apple One Individual plan"}]}
```


4.4.2.2 Disambiguation Dataset

In our unit dictionary, we encountered 18 ambiguous surface forms with different normalized units. To train the unit disambiguation module, a dataset of these 18 ambiguous surface forms was created, using ChatGPT. [9] The number 18 is not absolute and in different scientific domains, more ambiguous cases might occur. For each ambiguous surface form, at least 100 examples are generated, and the final training dataset consists of 1,835 sentences with various context information. Each sample consists of a sentence with an ambiguous surface and annotation for the correct unit. For more challenging surface forms, more samples are generated. The number of samples per surface form and associated units for each surface form are shown in Table 4.6.

A test dataset is generated in the same manner using ChatGPT, consisting of 180 samples, 10 samples per surface form.

We experimented with multiple prompts, using ChatGPT. The aim was to create training/test data in JSON format, where the sentences are not duplicates or too simple. For this purpose, two sentences were formulated (one for each unit, in each surface form) and are used as input examples of different contexts. The prompt explicitly asks for JSON format output and 20 samples, due to the

[9] ChatGPT from Openai: https://chat.openai.com/ (last accessed: 02.05.2024)

Table 4.6: Ambiguous surface forms, units associated with them, and the number of samples in the training set for each surface form and unit pair.

Surface Form	Normalized Units	# samples
c	cent, celsius	144
¥	Chinese yuan, Japanese yen	100
kn	Croatian kuna, knot	116
p	point, penny	149
R	south african rand, roentgen	100
b	barn, bit	127
’	foot, minute	104
′	foot, minute	104
"	inch, second	112
”	inch, second	112
C	celsius, coulomb	116
F	fahrenheit, farad	100
kt	kiloton, knot	100
B	byte, bel	107
P	poise, pixel	102
dram	armenian dram, dram	180
pound	pound sterling, pound-mass	131
a	acre, year	113

sequence length limitation of ChatGPT. The final prompt is shown below. `UNIT1` and `UNIT1` are replaced with different normalized units that share a surface form and `"SURFACE_FORM"` denotes this ambiguous surface form. `"Sentence1"` and `"Sentence2"` use the units in their relevant contexts.

```
Create a training set of 20 samples, for  "UNIT1" and "UNIT2", where in the text the
surface form of the unit is always "SURFACE_FORM", but the unit is different. Output in
JSON format as follows:

{"text":"Sentence1", "unit": "UNIT1" },
{"text":"Sentence2", "unit": "UNIT2" }
```

The sentences are manually checked for coherence and correctness before adding to the dataset. Incorrect samples with wrong units were removed. In some cases, surface forms were manually altered to match the specifications of the task. For certain units, multiple generations were required to get more complex sentences.

4.4.3 Implementation

CQE is implemented in Python 3.10. For dependency parsing, part-of-speech tagging, and the matching of rules spaCy 3.0.9 [10] is used. [11] The unit disambiguation module, with BERT-based

[10] SpaCy package: https://spacy.io/ (last accessed: 02.05.2024)

[11] The latest version of the repository is updated to a newer version of SpaCy, but to replicate the result mentioned in this section, version 3.0.9 should be fixed.

classifiers, is trained using spacy-transformers[12] for smooth intergeneration with other SpaCy modules. Each classifier is a `bert-base` model, trained for a maximum of 1000 steps with Adam optimizer and a learning rate of 0.001. For computation of the Levenshtein distance we utilized the `python-Levenshtein 0.25.1` [13] package and for the evaluation metrics we used `scikit-learn 1.0.0` [14].
Parsers were created to align the output format of different baselines so that the differences in output representation do not affect the evaluation. For instance, for IllQ, we normalize the scientific units. It is worth noting that despite the claim that currencies will be normalized in IllQ, manual normalization of a few currency units was also required. Another instance is Q3, where we account for differences in the representation of ranges and differences in the normalization of values, i.e., in the Q3 dictionary "minute" is normalized to "minute of arc". As a result, if a value is detected by a baseline but not standardized, or a unit is detected but not correctly normalized to the form present in the dataset, post-processing is applied for a unified output. In this way, a fair comparison among all models is ensured.

The goal of the post-processing is to achieve a unified representation, close to the benchmark dataset. Since baseline models have various definitions of quantities, and different unit dictionaries, it is expected that their output representation varies, although in many cases they refer to the same quantity. This is most prominent for the units, as different unit dictionaries or lack of unit normalization result in multiple surface forms for a single unit. For example, the unit "celsius" in the benchmark dataset is detected as "degree celsius" by Q3, "c" by R-Text, "c", "° celsiu" and"celsiu" by IllQ, "celsiu", "degrees celsiu" and "degree celsiu" by GPT-3, "degc" and "degC" by Grbd. As demonstrated in the example, GPT-3, Grbd, and IllQ use multiple surface forms for the same normalized unit. Although one can argue that multiple representations for the same unit point to a the lack of normalization altogether, we chose to map these variations to a single unified format. The only exception is for currencies in IllQ since their work explicitly claims that unit normalization is performed for currencies. For the entire post-processing steps, we refer to the evaluation repository.

To keep up with the recent trends in NLP and lack of a the baseline for concept detection, we introduce a large language model baseline with GPT-3. We choose GPT-3 due to accessibility and superior performance. [15] To tag sentences using GPT-3, we use the few-shot learning paradigm by prompting the model to tag quantities and units in the text, given 10 distinct examples. GPT-3 is mainly advertised as a task-agnostic, few-shot learners, and we have not performed extensive fine-tuning. With the 10 examples, we aim to account for a variety of outputs, i.e., compound units, when no quantity is present, noun-based units, and prefixes for scaling the magnitude of a value. Our full prompt is as follows, where the desired output is quantities represented in a numbered list, with change, value, unit surface form, unit, and concept.

[12] SpaCy Transformers package: `https://spacy.io/universe/project/spacy-transformers` (last accessed: 02.05.2024)
[13] `python-Levenshtein`: `https://pypi.org/project/python-Levenshtein/` (last accessed: 02.05.2024)
[14] `scikit-learn`: `https://scikit-learn.org/` (last accessed: 02.05.2024)
[15] At the time of conducting these experiments GPT-4 was not released and since OpenAI is costly repetition of whole experiments at this point with GPT-4 would introduce additional costs.

```
Tag quantities and units in the texts:
Sentence: Woot is selling refurbished, unlocked iPhone XR phones with 64GB of
storage for about $330.
Answer:
1. =, 1.64, GB, gigabyte, storage
2. ~, 330, $, dollar, iPhone XR phones

Sentence: The chain operates more than 600 supermarkets and less than 800  stores.
Answer:
1. >, 600, supermarkets, supermarkets, chain
2. <, 800, convenience stores, convenience stores, chain

Sentence: The spacecraft, which is about the size of a school bus, flew into Dimorphos
at a speed of about 4.1 miles per second, that’s roughly 14,760 miles per hour (23,760
kilometers per hour).
Answer:
1. ~, 4.1,  miles per second, mile per second, spacecraft
2. ~, 14760, miles per hour, mile per hour, spacecraft
3. ~, 23760, kilometers per hour, kilometer per hour, spacecraft

Sentence: And overnight dogecoin fell from 0.317 to 0.308, a 2.8 percent drop.
Answer:
1. =, 1.0.317-0.308, -, -, dogecoin
2. =, 2.8, percent, percentage, dogecoin

Sentence: This is about minus 387 Fahrenheit (minus 233 Celsius).
Answer:
1. ~, -387, Fahrenheit, Fahrenheit, -
2. ~, -233, Celsius, Celsius, -

Sentence: WhatsApp more than 2 billion users send  fewer than 100bn messages a day.
Answer:
1. >, 2000000000, users, users, WhatsApp
2. <, 100000000000, messages, messages, users

Sentence: This includes colors between red and blue - wavelengths ranging between 390
and 700 nm.
Answer:
1. =, 390-700, nm, nanometer, wavelengths

Sentence: You don’t have a two-year bachelor’s degree or a six to eight-year PhD degree.
Answer:
1. =, 2, year, year, bachelors degree
2. =, 6-8, year, year, PhD degree

Sentence: The price of CO2 and fuel
consumption are not clear.
Answer:
No quantities or units

Sentence:{sentence}
Answer:
```

`{sentence}` is replaced with a query sentence from the test set to be tagged.

Nevertheless, the output of GPT-3 is not deterministic and requires extreme post-processing. The post-processing includes cleaning the predicted values to include only numbers, normalization of the units even if the unit is misspelled, e.g., "celsiu" instead of "celsius", "ppb" to "parts-per-billion", or "€" to "euro". As a result, it cannot be deduced that GPT-3 can normalize units without additional post-processing and a dictionary of unit surface forms.
From the different model variations for generation, the *text-davinci-003* [16] model from the OpenAI API [17] is used with a sequence length of 512, temperature of 0.5, and no frequency or presence penalty (other parameters were left at the default setting). We are aware that with extensive fine-tuning and more training examples GPT-3 values are likely to improve. However, the purpose of this evaluation is neither prompt engineering nor designing training data for GPT-3, and the few short learning should suffice for a baseline.

4.4.4 Analysis of Results

Our system is compared against Q3, IllQ, R-Txt, Grbd, and GPT-3 on precision, recall, and F1-score for the detection of value, unit, change, and concept on both NewsQuant and R-Txt test datasets. Disambiguation classifiers are also compared with weighted micro-averaged precision, recall, and F1-score for unit classification against Q3. Permutation re-sampling is used to test for significant improvements in F1-scores (Riezler and III, 2005), which is more statistically coherent in comparison to the commonly paired bootstrap sampling (Koehn, 2004). In the following, we first describe the evaluation metrics used in this study and then present the results.

4.4.4.1 Metrics

Detection results are reported with precision, recall, and F1-score. These measures are common both for information retrieval and classification settings. All these metrics are built on four concepts:

- True Positive (tp): The predicted and the ground truth conditions are both true.
- False Positive (fp): The predicted condition is true but the ground truth condition is false.
- True Negative (tn): The predicted condition and the ground truth condition are both false.
- False Negative (fn): The predicted condition is False but the ground truth condition is true.

Precision computes how many retrieved/classified documents are correctly identified. Recall computes the number of relevant/correctly classified items among all the positive predictions or in other words, a fraction of all relevant items that were found.
These concepts are best explained for both retrieval and classification settings in Figure 4.3, where

[16] This was the latest model at the time of conducting these experiments. Due to the rapid change in this domain, this will not be the case at the point of reading this thesis.
[17] https://platform.openai.com/ (last accessed: 02.05.2024)

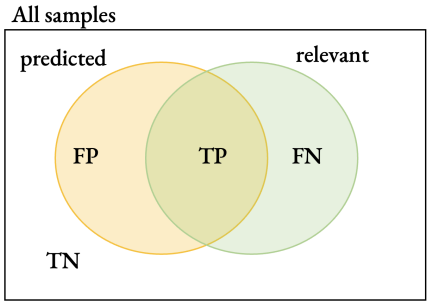


Figure 4.3: True Positive, False Positive, True Negative, and True Negative, where the green area is the ground truth relevant or positive samples and the yellow area is the model prediction.

the predicted condition in the case of a classification is the class of an element and for retrieval is the relevancy of an item.
Formally Precision (P) and Recall (R) are defined in Equation 4.1, where recall is also known as true positive rate.

$$P = \frac{tp}{tp + fp}, \quad R = \frac{tp}{tp + fn} \tag{4.1}$$

Precision and recall are complementary and are often reported together. If only one is reported the values can be misleading, for example, a recall of 1 is easily achievable by returning an entire corpus or classifying everything as positive, however, the precision in this case is abysmal.
To combine precision and recall into a single metric, the F1 score is introduced. The F1 score evaluates how well a model balances precision and recall, calculated as the harmonic mean of these two measures. As shown in Equation 4.2, the parameter β defines the relative importance of precision and recall. The common value of β is 1.

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \tag{4.2}$$

All the mentioned metrics can be averaged in case of multi-class classification, by taking the *macro*, *micro*, or *weighted* average. In such cases, we have per-class scores that need to be averaged to a single score, and the number of samples in each class plays a role in the final result. Often the dataset is imbalanced and the number of samples per class varies.
Macro averaging is computed by taking the arithmetic mean or unweighted mean of all per-class scores. This method treats all classes equally regardless of how many samples per class are available in the test set. If the dataset is imbalanced, this is not a good averaging technique.
Micro averaging computes a global average by counting the sum of tp, fn, and fp across all classes and then computing the respective metric. This metric is better suited to deal with class imbalance.
The weighted averaging is calculated by taking the mean of all per-class scores and taking the number of samples in each class into account. The scores are weighted, where a weight refers to the proportion of each class's support in the dataset.

Table 4.7: Precision, recall, and F1-score for detection of value, unit, and change on NewsQuant. Results with † mark highly significant improvements over the best-performing baseline with a p-value < 0.01.

Model	Value			Value+Unit			Value+Change		
	P	R	F1	P	R	F1	P	R	F1
CQE	**92.0**	**91.9**	**92.0**†	**85.6**	**85.5**	**85.6**†	**88.2**	**88.1**	**88.1**†
Q3	65.0	83.3	73.0	42.1	53.9	47.2	-	-	-
IllQ	50.6	66.0	57.3	32.8	42.8	37.1	44.2	57.6	50.0
R-Txt	59.7	82.2	69.1	29.6	40.7	34.2	-	-	-
Grbd	58.8	53.1	55.8	37.4	33.7	35.5	-	-	-
GPT-3	72.1	69.1	70.6	60.3	57.9	59.1	53.1	50.9	51.9

4.4.4.2 Results on NewsQuant

Table 4.8 shows the result on the NewsQuant dataset. Since Q3, R-Txt, and Grbd do not detect changes, respective entries are left empty. CQE beats all baselines in each category by a significant margin, where most of the errors come from the independent dependency parser and POS-tagger (SpaCy) and not the rules, i.e., mainly incorrect extraction of the dependency parsing tree and part-of-speech tagging. One of the common mistakes is when date and year information is wrongly tagged as a quantity by the POS-tagger, which increases the false positive rate.

The second-best model, Q3, scores highly for value detection but ignores all the noun base units. Q3 tends to overgeneralize tokens to units where none exist, e.g., in "0.1 percent at 5884", Q3, detects "percent per ampere-turn" as a potential unit, due to "at" being a surface form for "ampere-turn". Moreover, Q3 makes numerous mistakes on currencies and their normalization. We attribute this to their incomplete unit dictionary. Since we utilized the Q3 units dictionary to create our own, we can confirm the lack of entries for scarce and rarely used units and currencies.

R-Txt works well for the quantity types that they have a dedicated model for, but all the other quantities are ignored or misclassified. Since the four quantity types considered by R-Txt touch only the tip of the iceberg of all possible quantity and unit combinations, it is expected that many quantities in NewsQuant go undetected by this model. Another point that makes the use of R-Txt cumbersome is that one has to manually select a quantity type for the R-Txt prior to extraction. Therefore, we needed to run all the available models to find a quantity, which not only increases the runtime but increases errors due to the miss-classification of quantity types.

IllQ has extremely poor performance on the NewsQuant dataset. Despite the fact that IllQ is the only model that also detects noun-based units, the recall is much lower than for other models. This behavior is surprising and rather unfortunate since this is the only model that has been repeatedly used in literature as the foundation for other systems. IllQ has trouble with compound units, e.g., "$2.1 per gallon". There also seems to be a bias towards tagging the word after a value as a unit, e.g., in "women aged 25 to 54 grew by 1%", *grew by* is the falsely detected unit. Since IllQ does not have a normalization module or a unit dictionary, these mistakes are not easily avoided. Although IllQ is

claimed to normalize currencies, in practice the normalization is limited and often currency symbols are not normalized. Moreover, trends are ignored by IllQ, and the model is biased to predict equality (=) for most changes.

The Grdb model typically identifies the correct tokens for values, but due to inconsistent standardization, the final output is mainly incorrect. Unit normalization is limited to a small subset of units, where percentages and compound units that consist of multiple words, e.g., "dollar per barrel", are mainly ignored. In general, the model is biased to predict a single word for a unit, and if the unit is comprised of multiple tokens it is not detected.

GPT-3 achieves a score close to Q3 for the detection of value/unit pairs and close to IllQ for changes. Nevertheless, these numbers are not calculated based on the raw output of GPT-3. Due to extreme hallucination, extensive post-processing of the output is required for evaluation, e.g., many of the values extracted were not actual numbers and units were not normalized. Moreover, GPT-3 often confuses value suffixes with units, e.g., "billion" or "million". Despite the normalization prompt, GPT-3 fails to normalize units and requires manual normalization for most detections. Both IllQ and GPT-3 require extensive post-processing for units and cannot easily be used out-of-the-box.

Table 4.8: Precision, recall, and F1-score for detection of value and unit on R-Txt datasets. Results with † mark highly significant improvements over the best-performing baseline with a p-value < 0.01.

Model	Detect	currency			dimension			temperature			age		
		P	R	F1	P	R	F1	P	R	F1	P	R	F1
CQE	Value	**82.6**	85.9	**84.2**	**85.5**	87.6	**86.5**	**94.3**	97.1	95.7	**91.3**	**95.5**	**93.3**
Q3		69.2	84.7	76.2	76.9	**93.4**	84.3	91.7	97.1	94.3	**91.3**	**95.5**	**93.3**
IllQ		65.5	70.6	67.9	65.3	77.7	70.9	88.9	94.1	91.4	65.4	77.3	70.8
R-Txt		67.4	**91.8**	77.7	73.6	90.1	81.0	91.9	**100.0**	**95.8**	77.8	**95.5**	85.7
Grbd		46.6	35.3	40.2	75.8	59.5	66.7	84.0	61.8	71.2	60.0	27.3	37.5
GPT-3		50.5	54.9	52.6	80.2	80.2	80.2	93.5	85.3	89.2	92.3	54.5	68.6
CQE	Value +Unit	**78.1**	**81.2**	**79.6**†	**78.2**	**80.2**	**79.2**	91.4	94.1	92.8	**91.3**	**95.5**	**93.3**
Q3		29.5	36.1	32.5	56.5	68.6	61.9	61.1	76.5	74.3	82.6	86.4	84.4
IllQ		41.8	41.6	45.1	43.4	52.1	47.5	30.6	32.4	31.4	42.3	50.0	45.8
R-Txt		46.7	63.5	53.8	44.6	54.5	49.1	**91.9**	**100.0**	**95.8**	70.4	86.4	77.6
Grbd		24.9	18.8	21.4	44.2	34.7	38.9	32.0	23.5	27.1	40.0	18.2	25.0
GPT-3		40.8	44.3	42.5	65.3	65.3	65.3	45.2	41.2	43.1	92.3	54.5	68.6

4.4.4.3 Results on R-Txt Dataset

Evaluation results on the four quantity types of the R-Txt dataset are shown in Table 4.9, where the CQE model once again outperforms all baselines on value and unit detection for all categories except for temperature. Nevertheless, for temperature, the R-Txt improvement over CQE is not statistically significant. The small size of the age and temperature dataset results in inconsistent significance testing. The closeness of results for value detection between models is due to the structure of the dataset. The R-Txt dataset does not contain diverse values and unit representation. Values are

Table 4.9: Error analysis of different extraction systems.

Mistake	Systems
Trouble detecting temperature types, e.g., "celsius" and "Fahrenheit" are both denoted as "degrees".	Q3, IllQ, GPT-3, Grbd
Dollar types are not identified, e.g., "Hong Kong dollar" and "new Zealand dollar" → "dollar".	Q3, IllQ, Grbd
Unit normalization does not work for the majority of the time.	IllQ, GPT-3, Grbd
Bias towards predicting "=" for changes.	IllQ, GPT-3
Cryptocurrencies and rare currencies are not recognized, e.g. "Bitcoin" or "Markka".	Q3
Units used in sports are not recognized, e.g., "ppg", "rpg", "apg".	Q3
Temporal values are mistaken as quantities, e.g., "2 pm."	R-Text, GPT-3, Grdb
Compound units are rarely found, e.g., "kph".	R-Text, GPT-3
Units in incomplete sentences are not recognized, e.g., "rmb 10" and "usd 20".	CQE
No distinction between "year" and "year of age".	CQE, Q3, GPT-3
Units are confused with concepts, e.g., "building rate of $80 per sq m" → "sq m" as a concept.	GPT-3
Low recall due to limited quantity types.	R-Text, Grbd
Detection of concepts where none exist.	CQE
Problem with correct standardization of values.	Grdb
Multi-word compound units and most percentage units are ignored.	Grdb
Unable to correctly distinguish different temperature units.	Grdb

floats, and the diversity of types like ranges, fractions, and non-quantities is negligible. The R-Text model, with dedicated models for each of these quantity types, detects the quantity for the specific type with high accuracy. However, after annotating the remaining quantities in the test set, it has a hard time distinguishing the correct type, making the R-Text model unsuitable for general settings.

4.4.4.4 Error Analysis on NewsQuant and R-Text Datasets

We analyzed the incorrect detection for the models and the common mistake patterns on the test datasets. For a more systematic overview of the common mistakes made by each system, refer to Table 4.10. This list is not comprehensive and only contains the most prominent patterns that were observed by inspecting the false positive and false negative detections.

4.4.4.5 Concept Detection

Table 4.10: Relaxed and strict matching, precision, recall, and F1-score for concept detection on the NewsQuant dataset. Results annotated with † mark highly significant improvements over the best-performing baseline with a p-value < 0.01.

Model	Relaxed Match			Strict Match		
	P	R	F1	P	R	F1
CQE	**76.2**	**76.1**	**76.1**†	**57.0**	**57.0**	**57.0**†
GPT-3	55.9	53.7	54.8	26.3	25.2	25.7

The results of the concept detection on the NewsQuant dataset are shown in Table 4.11. Due to the similarity between concept detection and temporal tagging, where spans of text are tagged as a

Table 4.11: Weighted micro-average precision, recall, and F1-score on the unit disambiguation dataset for CQE and Q3. Results with † mark highly significant improvements over the best-performing baseline with a p-value < 0.01.

Model	micro-avg P	micro-avg R	micro-avg F1
CQE	**89.9**	**89.4**	**88.1**†
Q3	57.33	57.78	54.46

temporal entity, we follow the approach of (UzZaman et al., 2013b) for evaluation. Accordingly, two types of matches, namely, *strict* and *relaxed* matches are compared. A strict match is an exact token match, where the entire concept, word by word, is detected by the model. A relaxed match is counted when there is an overlap between the system's and ground truth token spans. Based on the scores we observe that concept detection is harder in comparison to value and unit detection. GPT-3 also struggles with accurate predictions. It is possible that the subtle nuance of what constitutes a concept requires more training data for a language model to learn.

On the other hand, our approach for concept detection is limited to common cases and does not take into account the full complexity of human language, leaving room for improvement in future work. Moreover, in many cases, the concept is implicit and hard to distinguish even for human annotators, causing the lower inter-annotator agreement in this setting. We aimed to create an approach that is more recall-oriented, trying to capture as many concepts as possible, hence, the big gap between relaxed and strict matches. In CQE, candidates are presented in a hierarchical structure based on the priority of the rules, by choosing to ignore concepts further down the hierarchy the model can be adjusted to be restrictive and precision-focused.

4.4.4.6 Unit Disambiguation

CQE is compared against Q3 (the only other systems with disambiguation capabilities) in Table 4.12. Since the normalization of units is not consistent in the GPT-3 model and requires manual normalization, GPT-3 is left out of this study.

In CQE, for each ambiguous unit, a classifier is trained, yet they all work together in the final system. As a result, all 18 classifiers are then assessed within a single system, with test data combined into one unified test set rather than separated by ambiguous units. This approach is more realistic and mirrors the practical use of the model.

The results are averaged by weighting the score of each class label by the number of true instances. CQE significantly outperforms Q3 on all metrics, and it is easily extendable to new unit surface forms and units by adding a new classifier. Since the training data is generated using ChatGPT, a new classifier can be trained using our paradigm and data generation described in Section 4.4.2.2, making this module extendible to new units.

Other than the micro-averaged results, we also present a detailed evaluation of the disambiguation dataset, where precision, recall, and F1-score are computed separately for each ambiguous unit. For

Table 4.12: Precision, recall, and F1-score for unit disambiguation per class.

Class	CQE			Q3		
	P	R	F1	P	R	F1
knot	100	100	100	16.67	100	28.57
roentgen	100	100	100	44.44	80	57.14
barn	100	80	88.89	100	80	88.89
japanese yen	60	100	75	100	100	100
inch	83.33	100	90.91	77.78	70	0.7368
armenian dram	100	60	75	0	0	0
chinese yuan	0	0	0	100	100	100
byte	100	100	100	57.14	80	66.67
cent	83.33	100	90.91	0	0	0
croatian kuna	100	100	100	0	0	0
year	100	100	100	0	0	0
poise	100	100	100	100	100	100
south african rand	100	100	100	100	60	75
minute	80	80	80	60	90	72
bit	83.33	100	90.91	50	40	44.44
bel	80	100	88.89	100	100	100
kiloton	100	100	100	100	100	100
second	100	80	88.89	80	40	53.33
coulomb	100	100	100	100	100	100
dram	71.43	100	83.33	0	0	0
point	55.56	100	71.43	0	0	0
fahrenheit	100	100	100	100	100	100
celsius	100	90	94.74	0	0	0
pixel	100	100	100	80	80	80
pound-mass	100	100	100	83.33	100	90.91
pound sterling	100	100	100	100	80	88.89
foot	80	80	80	100	50	66.67
penny	100	20	33.33	0	0	0
farad	100	80	88.89	80	80	80
acre	100	100	100	0	0	0

each surface form, 10 examples are present in the test dataset and the results are shown in Table 4.13. We noticed that distinguishing between "Japanese yen" and "Chinese yuan " is partially difficult for the BERT-based classifier since both of them are currencies and are used in similar contexts. Another hard distinction is between "penny" and "point", since monetary values and the point in the stock market are used in similar contexts. In comparison, in Q3 certain normalized forms are almost never predicted, hence the multiple zeros in the evaluation results.

4.5 Community Support

In addition to quantitative evaluation, we created an online interface named *QuantPlorer* that can be accessed under `https://quantplorer.ifi.uni-heidelberg.de/`. The interface is built on top of CQE and allows for interactive experiments with the model through an intuitive web interface to demonstrate its extractive power and simplify the interaction with the framework. QuantPlorer

extracts quantities from unstructured text using CQE and enables users to interactively investigate and visualize quantities in text. It also supports the filtering of results based units, specific ranges of values, related concepts, and changes. Users can gain an overview of the quantities in a document by looking at the distribution of values associated with a specific concept or unit. They can also further explore the document by visualizing the frequency of different units and concepts mentioned in the text through a histogram. For a large number of documents, manual interactions with the web interface are eliminated by using a programmatic API. [18]

4.5.1 Interface implementation

The backend is the CQE package, as described in Section 4.4.3. The front-end is a web application, which is served via Flask Python web server. [19] Since the input of CQE is sentences, before passing an input text to CQE, documents are split into sentences using the SpaCy library and passed independently to the extractor. The design is done using HTML, Javascript, and the Bootstrap 5 library [20] is used for a responsive layout. Charts and plots are generated using Chart.js [21] and the range slider comes from the open-source library of range-slider-input. [22] Communication between the interface and the server is based on AJAX and JSON objects are used to pass information in both directions from the server and back.

4.5.2 Demonstration

We motivate the use case of CQE for the exploration of quantities in unstructured documents by going over typical workflows that Quantplorer presents. Although the workflows described here are limited to a single document, applying the extraction on large corpora and aggregating the statistics can provide a more comprehensive overview. As mentioned before, for a large number of documents the use of the programmatic API is advised. For the examples presented, we compiled relevant paragraphs on a specific topic, i.e., Google's earnings across different sectors, into a single document. In the following, workflows for (1) inspecting the extractions, (2) filtering the data based on quantity attributes, and (3) gaining an overview of the document by looking at statistics of values is presented.

Extraction: The entry point to the interface is the extraction editor, which is shown in Figure 4.4. The user can either explore the provided examples or add a new document by clicking on the "+" symbol. The text editor is filled either by dragging a text file or directly typing in the editor.
By clicking the *Extract* button, all quantities and their respective components (value, unit, change, and concept) are extracted and color-coded within the text for easy identification.

[18] Quantplorer API : https://quantplorer.ifi.uni-heidelberg.de/api/extract (last accessed: 02.05.2024)
[19] Flask framework: https://flask.palletsprojects.com/en/2.0.x/ (last accessed: 02.05.2024)
[20] Bootstrap: http://getbootstrap.com (last accessed: 02.05.2024)
[21] Chart JS:https://www.chartjs.org/ (last accessed: 02.05.2024)
[22] Input-sider repository:https://github.com/n3r4zzurr0/range-slider-input (last accessed: 02.05.2024)

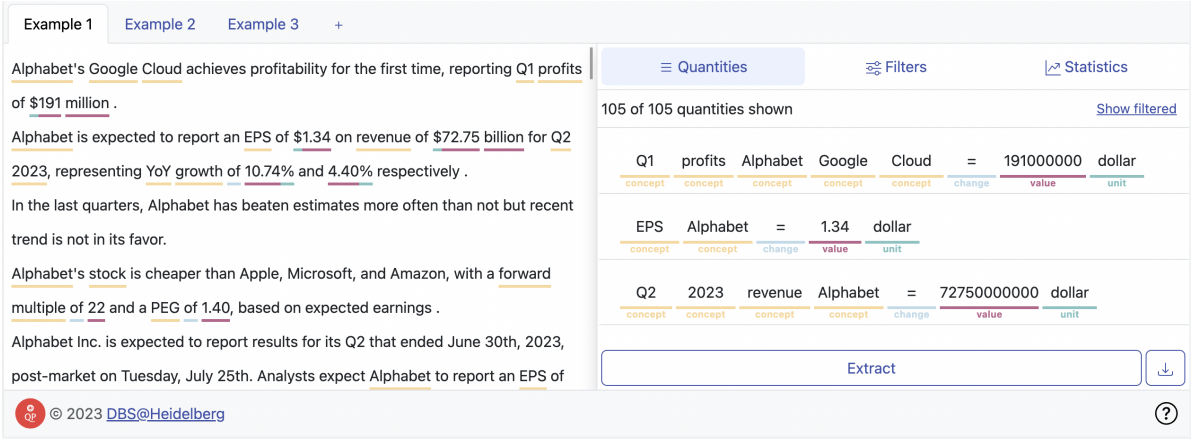


Figure 4.4: The entry point to Quantplorer. The interface contains a text editor, where the quantities and their components are color-coded and the standardized values and normalized units are shown in the right sidebar.

The *Quantities* tab under the right sidebar in Figure 4.4 contains the extractions in a list format. The surface forms of units and values are highlighted with an underline in the text editor and the standardized values, normalized units, and changes are displayed in the extraction block. By clicking on an extraction, the source sentence of the extraction is highlighted for more clarity. The highlighting and color coding in the editor provides an immediate overview of quantities in a document.
The download button next to the bottom right corner can be used to download extraction results for a single document in JSON format.
The question mark on the bottom right corner of the screen opens a helper page that presents the user with more information about the framework and examples on how to use the API.

Filtering: The *Filters* tab allows the user to view and download annotations for quantities that match certain criteria. The filtering options are selected based on the quantity attributes extracted from the CQE. Users can specify the following filters:

- ranges of values,
- specific concepts,
- restriction on changes,
- specific units,
- other keywords from sentences in the document.

Upon choosing a filter, the *Quantities* tab is automatically updated to reflect the selections. As a result, the download button will then export only the quantities matching the filters.
The concepts are usually multiple words, and for filtering they are presented as uni-grams. Therefore, if the user is looking for all sentences with a multi-word concept, e.g., “Alphabet revenue”, both

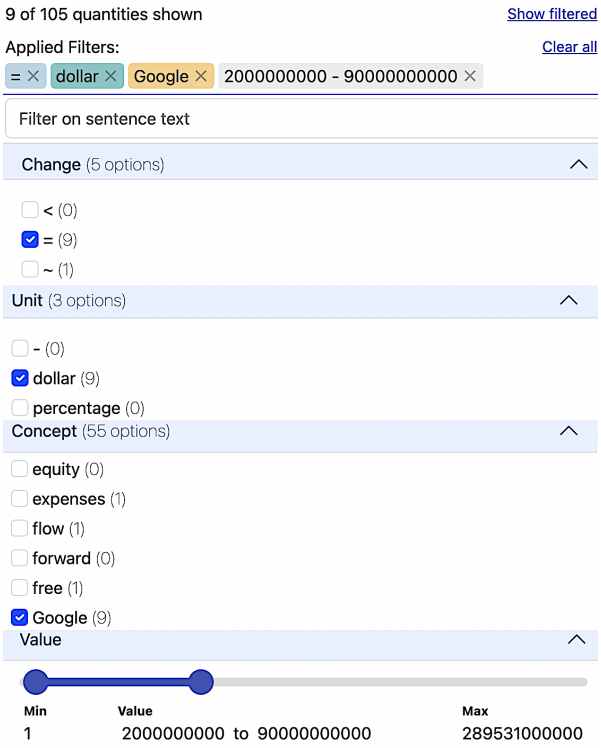


Figure 4.5: Filtering options based on the quantity components extracted from CQE.

uni-grams need to be selected individually.

An example of the filtering option is shown in Figure 4.5. The numbers in parenthesis show the number of quantities that have this particular attribute under the given filters. For example, in Figure 4.5 with the current filters, there are no quantities that contain the concept "equity" but a single one containing the concept "expenses".

The value slider displays the minimum and maximum values among all quantities in the text, irrespective of the unit. Users can adjust the slider to set their desired filtering conditions.

Each filter appears as a closable span above the list of annotations, allowing users to remove filters by closing the span. The total number of filtered quantities is displayed at the top of the page. As shown in the example in Figure 4.5, 9 out of 105 quantities contain the concept "Google" and the unit "dollar", where the values lie between 2 and 90 billion and the change is equal.

Finally, the text box containing the text "Filter on sentence text" is used for keyword search within the document.

Statistics: One way to summarize a document rich with quantities is to look at the distribution of values associated with various concepts or units. Another way is to look at the frequency of various units and concepts, which provides a rough idea of the topic of the documents and the type of quantities mentioned in the text. The *Statistics* tab provides such functionalities, through (1) *Unit distribution*, (2) *Concept Distributions*, and (3) *Value Distribution*.

Unit distribution and *Concept Distributions* illustrate the frequency of distinct units and concepts in

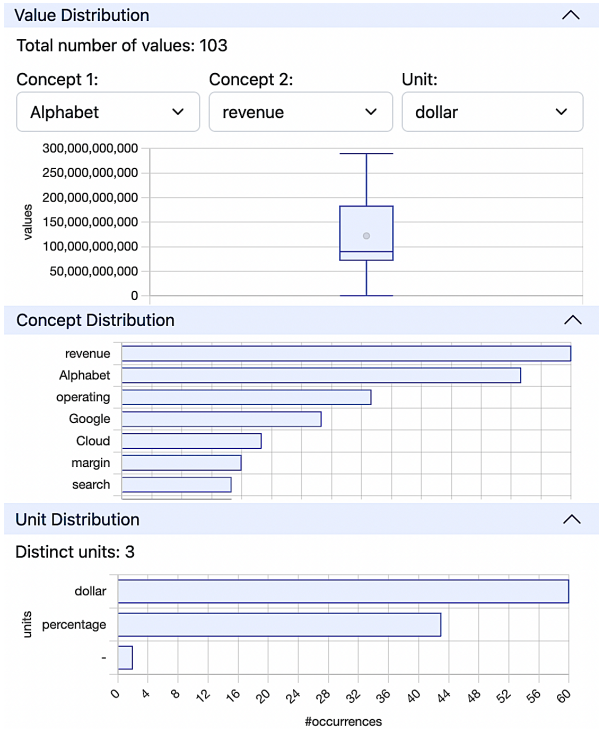


Figure 4.6: Statistics tab with the distribution of values, concepts, and units.

a given document using histograms. The user can understand the frequent types of quantities and concepts revolving around values by looking at these histograms.

An example of the *Statistics* tab is shown in Figure 4.6. In this case, the example document is dominated by the units "percentage" and "dollar". Moreover, quantities mainly revolve around the concepts "Alphabet" and "revenue".

The *Value Distribution* tab presents a fine-grained analysis, where the user can choose a single or combination of two concepts with a specified unit to analyze the distribution of values. In the example in Figure 4.6, the user can see the distribution of "Alphabet revenue" in "dollars" to get a sense of ranges and scales of values. Since the paragraphs chosen for this example show the revenue of Alphabet from 2019 to 2023, we can see that in general the revenue for these years is expected to lie between 50 and 150 billion. Although such information for companies and the stock market is easily found on spreadsheets and financial reports, it is important to note that this information is extracted from inspecting unstructured text. In domains where such information is not recorded separately, extraction of quantities and related concepts can help to gain insights previously unknown.

API: To quickly obtain extractions for a collection of documents without the need to install the CQE package or interact with the web interface, we additionally provide an API endpoint that returns extractions in JSON format.

4.6 Summary and Discussion

In this chapter, we defined a unified representation for quantities in text and contextualized information surrounding them. To extract such information from unstructured text, we introduced a comprehensive quantity extractor, based on a set of rules from dependency parsing tree and POS-tags. Our extractor is capable of standardizing values, normalizing units, and finding contextualized information in the form of change and concept from a sentence. CQE was compared on a novel dataset against available liberalities for quantity extraction and outperformed all baselines by a high margin. Finally, for community support and accessibility of the extractor we designed and deployed a web interface to explore the extraction of quantities interactively. In the following, we point out the limitations of CQE, future work, and the effect of large language models on quantity extraction.
Despite an extensive effort to account for most common cases, CQE is still mainly a rule-based approach, requiring manual feature engineering and rule-writing for unseen cases. This issue is more prominent in the case of concept extraction, where the order in which we apply the rules has a direct impact on correct extractions. If the rule with higher priority finds a candidate, the rules further down the list are ignored. Although for humans identifying the correct rule to use is easy by considering context and sentence formulation, such delicate differences in language are not easily captured in rule-based systems. However, to the point that was possible, we focused on extendibility and flexibility. For example, upon the addition of a new entry to the unit dictionary, it will automatically be detected and normalized. This flexibility extends to linguistic rule sets as well. Adding and removing a rule will automatically change the behavior of the system.
Another source of error is that CQE relies heavily on correct dependency parsing and POS tagging, and any error on the initial extraction propagates through the entire system. Consequently, even changes in the versions of the SpaCy model used for dependency parsing and POS tagging can produce slightly varying results.
Given the rapid evolution of large language models, our comparison to GPT-3 may not accurately reflect the current capabilities of these models in quantity extraction. At the time of the experiments, GPT-3 was the state-of-the-art model for large language modeling. Consequently, the performance of newer models like GPT-4, BART, and Gemini on our benchmark remains unknown. In our experiment with GPT-3, one major drawback was the variability in the output. Beyond formatting issues, GPT-3 does not provide concrete and stable functionality like the other baselines. For instance, it may opt to standardize a value or not, normalize a unit or not, introducing unpredictability. This behavior makes it hard to use such models, where a deterministic output is desired.

5 Quantity-centric Ranking

> "We are drowning in information, while starving for wisdom."
>
> *E.O. Wilson*

Armed with methods to extract quantities from unstructured text, we can set out to investigate ways to incorporate quantity understanding in information retrieval systems. Despite advances in semantic search and sophisticated neural network architectures, handling quantitative information in text remains challenging, specifically performance on quantity-centric queries, in which the query contains a quantity and a numerical condition, is low, e.g., "BMW with more than 530hp". The reason for this is that systems are not aware of numbers and their semantics, such as proximity, in particular in combination with units. Numbers and units are treated in the same way as any other text token that is subject to subsequent processing, i.e., indexing or embedding. What complicates things further is that numbers and units can also have different surface forms (e.g., "6k" vs "6,000" and "mph" vs "miles per hour") and require standardization and normalization (Weikum, 2020). While there are approaches that specifically focus on numbers in text, i.e., extracting quantities for entities (Ho et al., 2019; Li et al., 2021), linking quantities in tables (Ibrahim et al., 2019), or numerical reasoning (Ran et al., 2019), they are tailored for specific tasks and not semantic search in general. This applies to neural models supporting IR, which are trained on general-purpose data without the focus on quantity semantics. As mentioned in related work in Section 3.3, language models, which modern retrieval systems are based on, exhibit a limited understanding of number scales and proximity (Wallace et al., 2019). The fundamental assumption of language models is that the semantics of a word is inferred from the company it keeps (Firth, 1957), which is not necessarily true for numbers. The context of a value may offer some insight into valid ranges, but this becomes uncertain when units are in play. For instance, reporting a company's revenue in different currencies can result in significantly varying ranges, despite the context roughly talking about the same topic. Recent works on numerical language models (Spokoyny et al., 2022; Jin et al., 2021) focus on better numerical representation. Yet, these architectures are very specific to certain downstream tasks and require changes in the architecture of popularly used language models in IR, which indicates expensive pre-training. As a result, quantity understanding in retrieval systems remains an uncharted area, in particular numerical conditions and their semantics, as discussed in Section 3.2, is widely under-

studied. The main body of work in this area is in the context of QSearch (Ho et al., 2019, 2020), where the focus is narrowed down to named entities. In reality, quantities are related to concepts that go beyond the narrow focus of named entities, e.g., "square footage of a house", which makes such approaches tailored to specific use cases and not general settings.

Current neural retrieval models are trained on general-purpose queries without the focus on quantity semantics, where a small portion of the queries containing quantities is either focused on a reasoning task or retrieval of a single value related to a concept, e.g., "height of Eiffel tower". Answering queries of these types is possible with span detection and without the need for quantity understanding. Since these models never had task-specific training, are based on language models with little understanding of scales, and do not have a separate ranking module for quantity comparison, it is no wonder that they perform poorly on such a task (as shown later in our evaluation). Moreover, the lack of available benchmark datasets for training or comparison of quantity-centric queries is exacerbating the problem.

In this chapter, we present two strategies to enhance the quantity understanding of current IR systems. We aim for a general-purpose model that is not specific to quantity ranking but is also capable of term-based ranking. We specifically focus on queries containing quantities and numerical conditions. [1] The quantity-centric ranking follows the same intuition as described in Chapter 2 for general retrieval, where the relevance is computed as a function of the query and document similarity, $r(d_i|x) \sim sim(\tau(x), \tau(d_i))$. τ is a generic mapping function that converts the query x and document d_i to their dense or sparse representations. We focus on sentences as basic retrieval units. Hence, instead of relevance to the document d_i, we focus on relevance to the sentence s_i and an estimation of $r(s_i|x)$.

In the context of quantity-centric ranking, $r(s_i|x)$ is dependent not only on the search terms but also on the query quantity and numerical condition. When considering a quantity, both the value and unit carry significance. For instance, in a query like "iPhone with price more than €800", the quantities in the highly ranked sentences should have Euro as a unit. Moreover, the same search term and quantity may be deemed relevant or irrelevant depending on the numerical condition. For example, the sentence "iPhone X costs €999" is relevant for the query "iPhone price more than €800" but irrelevant to the query "iPhone price less than €800".

Throughout this chapter, we assume that our retrieval system is operating on a document collection where each document consists of a sequence of sentences, which may or may not contain quantities. The methods proposed in this chapter focus on four main principles:

1. *Versatility:* The proposed methods should be applicable in real-world scenarios and not limited to an academic setting or a specified data type. Specifically, a quantity-centric ranking should not hinder a model's capability to do traditional term-based ranking.

[1] It can be argued that queries that contain numerical reasoning and aggregation of values, e.g., "What is the average profit of Apple Inc. per year?", are also categorized as quantity queries. Nevertheless, our emphasis is directed specifically towards the scenarios containing numerical conditions.

2. *Efficiency:* An essential part of retrieval systems is efficiency and response rate, and the same should apply to quantity-centric ranking.

3. *Generalizability:* It is important that the proposed model is generalizable to different application domains with minimal effort.

4. *Applicable to low resource domains:* As mentioned previously, there exists no open source dataset for training quantity-centric retrieval models, and thus, the proposed approach should be able to work in low resource settings.

Taking into account these four principles, we present two approaches to enrich current retrieval systems with quantity understanding. We aim for a general-purpose model that is not specific to quantity ranking but is also capable of term-based ranking. The two approaches differ in their integration of quantity ranking with term-based ranking. The first employs a disjoint combination of a quantity ranking score with term-based ranking, while the second focuses on the joint ranking of quantities within the context of textual content.

A note on the terminology used in the following: Here, we refer to term-based ranking as both semantic search and lexical-based models. The lexical model and keyword-based search are used interchangeably. Semantic models include neural models that can have both dense and sparse representations. Moreover, the task that we focus on is ranking, where a ranking function outputs a score that represents the relevance of a document or sentence to a given query.

Contributions. In this chapter, we make the following five contributions:

1. We propose a disjoint quantity-centric ranking approach. This model is unsupervised, utilizing an index structure, and compatible with various lexical and semantic IR systems.

2. Due to the independence assumption of the disjoint model, the relationship between quantities and surrounding text is to some extent lost. Therefore, we propose a joint quantity ranking model. In this case, we aim to learn quantity-aware sentence and query representations through task-specific fine-tuning of neural IR models.

3. We introduce two novel benchmark datasets for quantity-centric ranking, specifically focusing on queries involving numerical conditions in the domains of finance and medicine.

4. We evaluate the performance of our systems against various lexical and neural models and show significant improvements over these baselines.

5. Finally, we investigate the effect of quantity-centric retrieval in modern question answering systems based on the Retrieval Augmented Generation (RAG) paradigm.

References. Parts of this chapter are based on these publications:

Satya Almasian, Milena Bruseva, and Michael Gertz. QFinder: A Framework for Quantity-centric Ranking. In *SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022*, pages 3272–3277. ACM, 2022b. URL https://doi.org/10.1145/3477495.3531672

Satya Almasian, Vivian Kazakova, and Michael Gertz. Numbers Matter! Bringing Quantity-awareness to Retrieval Systems. Under Review, 2024a

Structure. We start by describing the disjoint quantity-centric ranking in Section 5.1. The combination of unsupervised and heuristic quantity ranking with lexical and semantic rankers is described in Section 5.1.3. We move on to joint quantity-centric ranking with supervised approaches in Section 5.2 and propose a semi-supervised fine-tuning paradigm with concept expansion, and value and unit permutations. Section 5.3 describes the effect of quantity understanding in retrieval augmented generation pipelines, where the query contains numerical conditions. Section 5.4 introduces the two benchmark datasets, FinQuant and MedQuant for the evaluation of quantity-centric systems and compares the performance of the proposed models against various baselines. Finally, in Section 5.5, an interactive interface for the disjoint quantity ranking model is demonstrated for further exploration.

5.1 Disjoint Ranking of Quantities and Terms

The disjoint model is based on the separation of quantity and term ranking. We assume that the term-based relevance of a sentence to query terms is independent of the proximity of query and sentence values under the query condition.

Definition 5.1. *(Quantity-centric Query).*
Given a sentence $s_i = (T_i, Q_i)$ *as a sequence of tokens* $T_i = (t_1, ..., t_l)$ *and quantities* $Q_i = (q_1, ..., q_k)$, *where a quantity* $q_i = (u_i, v_i)$ *is a tuple of a unit* u_i *and a value* v_i.
A quantity-centric query is given by $x = (T_x, c, q_x)$, *where* $T_x = (t_{x_1}, .., t_{x_n})$ *are the search terms related to the query quantity* $q_x = (u_x, V_x)$.
The query value is a pair of two numbers $V_x = (v_{i_1}, v_{i_2})$, *such that the ranges are considered, with* v_{i_1} *being the lower bound and* v_{i_2} *the upper bound of a range. For all other conditions that require a single value for comparison,* v_{i_2} *is set to undefined, i.e.,* $V_x = (v_{i_1}, \perp)$.
c *is a numerical condition from the set* $\{=, <, >, [\,]\}$ *defining equal, less than, greater than, or range of values, respectively.*

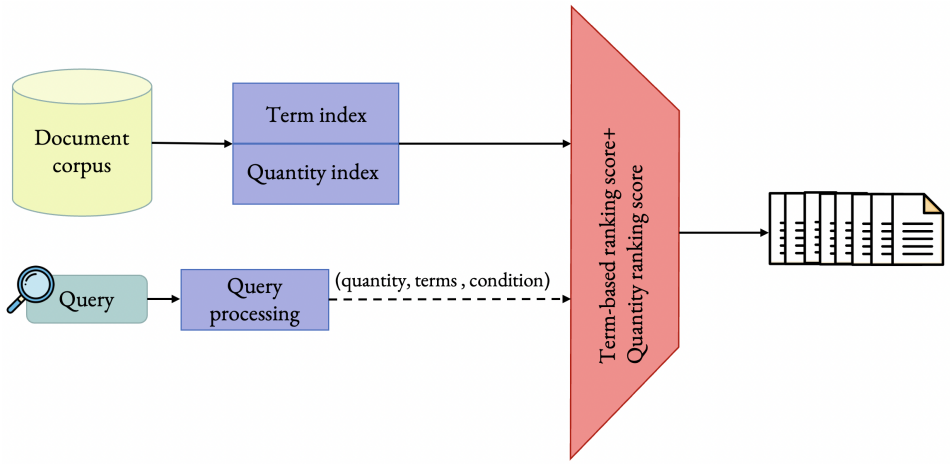


Figure 5.1: Pipeline of the disjoint quantity-ranking approach, where a separate quantity index is used to compute quantity proximity and a term-based lexical or semantic index is used to compute the similarity of the search term to the candidate sentences.

Definition 5.2. *(Disjoint Quantity-centric Ranking).*
Given a quantity-centric query as defined in Definition 5.1, the relevance, $r(s_i|x)$, of a sentence s_i to a query x in a disjoint quantity-centric ranking model is defined as

$$r(s_i|x) \sim sim(\tau(T_x), \tau(T_i)) + \alpha sim_c(\tau^{'}(q_x), \tau^{'}(Q_i)).$$

sim computes the similarity of search terms (T_x) to sentence terms (T_i) independent of sim_c, which computes the proximity of query quantity (q_x) and sentence quantities (Q_i) given the query condition c. The parameter α controls the contribution of quantity proximity to the final ranking. τ and $\tau^{'}$ denote that representations for query and document, respectively. These representations do not need to be based on the same model.

If a query is not quantity-centric, simply by removing the quantity score (sim_c), the models fall back to term-based ranking. The quantity-aware ranking model proposed in the remainder of this section is a set of heuristic ranking functions that do not require training data. The choice of a term-based ranking function is independent of the quantity-ranking, allowing for more flexibility and the ability to combine with the variety of term-based retrieval models.

The pipeline for the disjoint quantity-ranking model is shown in Figure 5.1. The query is processed into quantity, search terms, and a query condition, using CQE, described in Chapter 4, or similar packages. Our ranking approach consists of two parts: first, a term-based ranking function, and second, a quantity raking function for numerical proximity using a *quantity-index*.
The document corpus is indexed separately for terms and quantities. The term-based index can be a traditional lexical index or a vector database that retrieves semantically similar text chunks. On the other hand, the quantity index facilitates (1) finding the values that share the same unit as the query

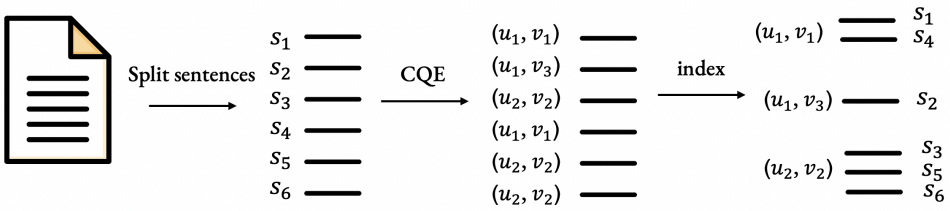


Figure 5.2: Quantity index creation steps, including sentence splitting, extraction of value/unit pairs using CQE, and creating a posting list for each distinct occurrence of value/unit pair.

and (2) computing proximity based on the query condition. The final ranking combines scores from term-based and quantity ranking.

In the following sections, we first describe the pre-processing and quantity index creation. We then delve into the heuristic quantity ranking function ($sim_c(\tau'(q_x), \tau'(Q_i))$). Finally, we describe how to combine the quantity score with a variety of term-based ranking methods for the final ranking.

5.1.1 Pre-processing and Quantity Index Creation

For the computation of the quantity score, a separate index for quantities is created, which provides explicit knowledge about values and units in normalized form. The index structure mirrors term-based indices used in lexical-based retrieval systems, with each value and unit pair containing a posting list of documents where it occurs. Leveraging such an index, we define heuristic functions that compute the proximity of values based on different numerical conditions.

Before index creation, pre-processing is performed, involving sentence splitting and extracting quantities from the corpus. Figure 5.2 illustrates the pre-processing steps and the indexing process for an example document containing six sentences. To extract quantities CQE package is utilized. [2] As the retrieval units and input for CQE consist of sentences, the first step is to split the input corpus into individual sentences. Subsequently, each sentence is processed independently by CQE. From the contextualized quantity tuple returned by the CQE extractor (value, unit, change, concept), concepts and changes are disregarded. [3] Through CQE, values are standardized, e.g.,"$300 million" to "$300,000,000", and units are normalized, e.g., "kilometer per hour" instead of "km/h'. The quantity index is built using unique pairs of normalized units and standardized values extracted from the corpus. Each pair references the sentence or list of sentences where that specific quantity appears, analogous to inverted indices for terms. If a sentence contains multiple quantities with the same unit, the quantity index will have multiple entries to reflect the frequency.

An example of such an index for three value/unit pairs is shown in Figure 5.3. The values in the

[2] The system is not dependent on the use of CQE and any other extractor that provides standardized values and normalized units can be used.

[3] One could argue that considering the changes is important for the correctness of retrieval systems, e.g., "he had more than 10 dollars", which means he does not have exactly 10 dollars. Our index structure currently does not differentiate between such subtleties, which gives rise to possible future work in this direction.

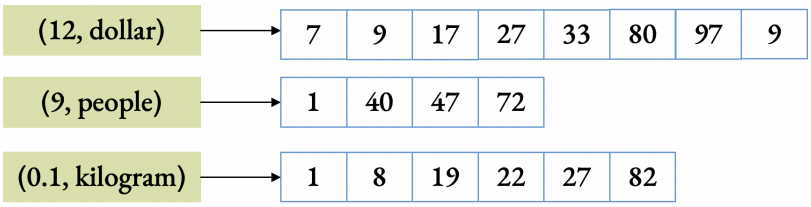


Figure 5.3: An example of quantity index, where value and unit pairs point to posting a list of document IDs.

posting list are the sentence IDs from the corpus. It is important to note that quantity values are in standard decimal format, and units are normalized.

5.1.2 Quantity Ranking Functions

To estimate the numerical proximity ($sim_c(\tau(q_x), \tau(Q_i))$ as defined in Definition 5.2 and to rank the quantities in sentences (Q_i) based on the query quantity (q_x) and condition (c), we use a heuristic function. We call this ranking function $QScore$, which assesses the relevance of the query value to the value in a sentence under a given condition ($r(Q_i|q_x, c) := QScore(s_i, c, x)$). For higher values of $QScore$, the probability of relevance increases. This ranking function is dependent on the numerical condition, resulting in different scores for the same value under various numerical conditions. Additionally, the numerical score only matters if the units match, otherwise, the values are not comparable and refer to aspects of an object that are fundamentally different, e.g., "horsepower" of a car is different from "km/h" it reaches and should not be compared.

For a query quantity $q_x = (V_x, u_x)$ with condition $c \in \{=, <, >, [\ ]\}$, $QScore$ for sentence s_i is shown in Equation 5.1. The indicator function $\mathbb{1}_{u_i}(u_x)$ enforces the match between the unit in the query and the sentence, and Φ_c is the condition-dependent ranking function. To obtain a value between 0 and 1, the score is normalized by the number of quantities $|Q_i|$ in sentence s_i. In other words, the relevance of quantities in a sentence Q_i to the query quantity q_x under condition c is computed by the ranking function Φ_c, for quantities where the unit matches u_x.

$$\text{QScore}(s_i, c, X) := \frac{1}{|Q_i|} \sum_{i=1}^{|Q_i|} \mathbb{1}_{u_i}(u_x) \Phi_c(V_x, v_i) \tag{5.1}$$

Φ_c is a set of four functions, one for each condition $c \in \{=, <, >, [\ ]\}$. In the following subsections, the functions and the implications behind them are described.

All sentences containing quantities that share the unit of the query quantity u_x, where the values satisfy a numerical condition are considered relevant to the query. However, the order in which the results are presented to the user can either aid or hinder the user in finding the desired result. In term-based ranking, the optimal order of a result is evident. However, when it comes to quantities, ordering is more subjective and the optimal order is dependent on the user's information needs. For

example, a user searching for "iPhone display bigger than 8 inches" might look for a maximum value larger than 8 inches or a display only marginally larger, both of which are valid answers. However, if the results are presented in an ascending or descending order based on numerical distances, the user can identify the desired result much faster. This issue becomes more prominent in the case of ranges, e.g., "iPhone display between 8 to 12 inches", where it is not obvious if the user searches for values closer to the lower bound, the upper bound or perhaps near the average. For each ranking function, we discuss possible alternatives to achieve different sortings. The definition of a single optimal sorting of results and underlying ranking functions, respectively, is not necessarily meaningful and could be defined by the user's needs in a transparent fashion, e.g., the user can choose among ascending or descending numerical proximity during the search. An advantage of the disjoint approach is its inherent flexibility. Simply by switching between different variants of ranking functions, the results can be rearranged based on the desired sorting. Nonetheless, for the evaluation of the disjoint rankers against other baselines, we focus only on the most intuitive variant, which ranks quantities with values closer to the query value in descending order.

In the following, we describe variants of Φ_c for numerical conditions of *equal* ($\Phi_=$), *greater than* ($\Phi_>$), *less than* ($\Phi_<$) and *ranges* ($\Phi_{[]}$). For each function, we discuss variations that result in ascending, descending, and no sorting.

5.1.2.1 Equality Ranking Function

The equality ranking function $\Phi_=$ facilitates finding quantities that have values equal to or approximately equal to the query value.

No Sorting: This is the most restrictive function, without any sorting, where only the exact matches are ranked high. In this case, $\Phi_=$ assigns a score of 1 if $v_{x_1} = v_i$, and 0 otherwise, as shown in Equation 5.2. For the example query of "Car with maximum speed 245 km/h.", only the value exactly equal to the query value is considered valid and a sentence like "Audi RS6 maximum speed is 250 km/h." would receive a quantity score of zero, although the value is close to "245".

$$\Phi_= =: \begin{cases} 1 & v_{x_1} = v_i \\ 0 & else \end{cases} \tag{5.2}$$

Descending order: In this case, $\Phi_=$ determines the proximity of the query value v_{x_1} to the sentence value v_i based on the exponential decay of their difference, hereby, ranking closer quantities higher and in descending order, as shown in Equation 5.3.

$$\Phi_=(v_{x_1}, v_i) =: exp(-|v_{x_1} - v_i|) \tag{5.3}$$

The final score ranges between 0 and 1, with larger absolute differences yielding lower scores. For example, $\Phi_=$ for the sentence "Audi RS6 maximum speed is 245 km/h." with the query "Car max-

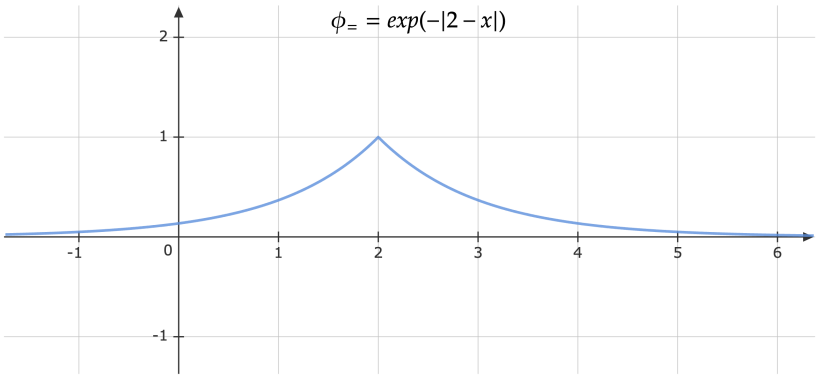


Figure 5.4: An example exponential decay for query value 2.

imum speed 250 km/h" is computed as $exp(-|245 - 250|) = 0.006$. In contrast, if the value in the sentence were 260, the score would diminish to $exp(-|245 - 260|) = 0.0000003$. An example plot of the ranking function concerted around the query value of 2 is depicted in Figure 5.4, where the score peaks at values equal to 2 and decays as the absolute distance increases. Note that the score exhibits a rapid decline to zero, where for values with greater distance to the query value, the score converges quickly to zero.

Ascending sorting: For the equality condition, the ascending sorting of results based on quantity proximity does not make sense. If the user is searching for values equal to the query values, sorting in ascending order, where the values further away from the query value are ranked higher, is not representative of the equality condition. Such sorting is more suitable for the *non-equality* condition, where the desired results should not contain the value in the query. However, we do not consider non-equality in this thesis.

5.1.2.2 Less Than and Greater Than Ranking Functions

$\Phi_{<}$ and $\Phi_{>}$ are used to find values less than and greater than a query value, respectively. In this section, we introduce (1) a ratio and (2) an exponential decay function for these boundary conditions, presenting arguments for the preference of the ratio function.

Ratio descending order: The numerical score in this case is defined by the ratio of a query value v_{x_1} and a sentence value v_i, resulting in a score between 0 and 1. The ratio determines the numerical proximity independent of the magnitude, where the score is higher for values with smaller differences. To account for the smaller value consistently appearing in the numerator and to maintain the ratio within the range of 0 to 1, if the query value v_{x_1} is greater than the sentence value v_i, the score is defined by the ratio of v_i to v_{x_1}, and vice versa. Since less than and greater than conditions define a

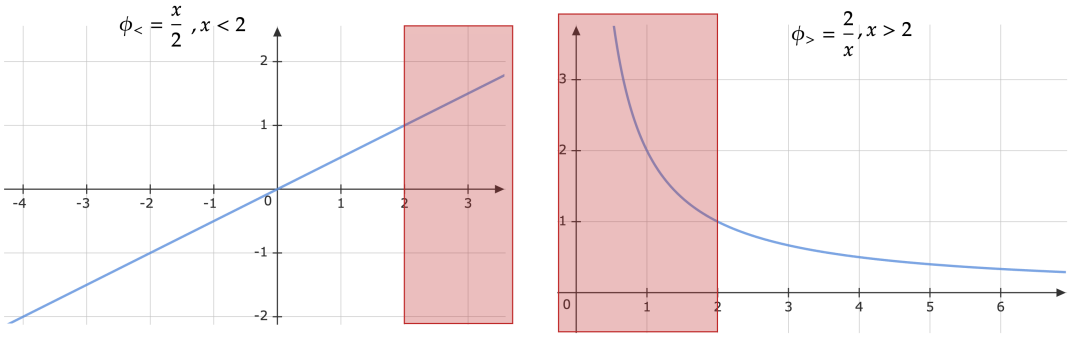


Figure 5.5: An example ratio ranking with descending order for less than (left) and greater than (right), with query value 2.

bound of valid ranges, any value that is strictly out of the bound is given the score of zero [4]. The ratio ranking for less than ($\Phi_<$) and greater then ($\Phi_>$) with descending order is shown in Equation 5.4.

$$\Phi_>(v_{x_1}, v_i) =: \begin{cases} v_{x_1}/v_i & v_{x_1} < v_i \\ 0 & else \end{cases}$$
$$\Phi_<(v_{x_1}, v_i) =: \begin{cases} v_i/v_{x_1} & v_{x_1} > v_i \\ 0 & else \end{cases} \tag{5.4}$$

Assume the query "iPhone XS $<$ 1500 dollars" and the two sentence with quantities q_1 = (1490, "dollar") and q_2 = (800, "dollar"). Based on the ratios $\frac{v_1}{v_{x_1}} = \frac{1490}{1500}$ and $\frac{v_2}{v_{x_1}} = \frac{800}{1500}$, q_1 receives a higher score as its value is closer to "1500".

One drawback of the ratio-based scoring is that the $\Phi_<$ is linear while $\Phi_>$ is convex, as shown in Figure 5.5 for an example query value of 2. This causes an unbalanced scoring between these two functions. Consequently, we also present an exponential decay variant of the functions, similar to the equality condition. However, in practice, the ratios produce a high-quality ranking and the computing and exponential function is more expensive.

[4] It is also possible to change the function to assign small scores to out of range quantities, however, this value has to be less than the score for values within the valid range. Estimation of such value is difficult and therefore, we keep the score of zero.

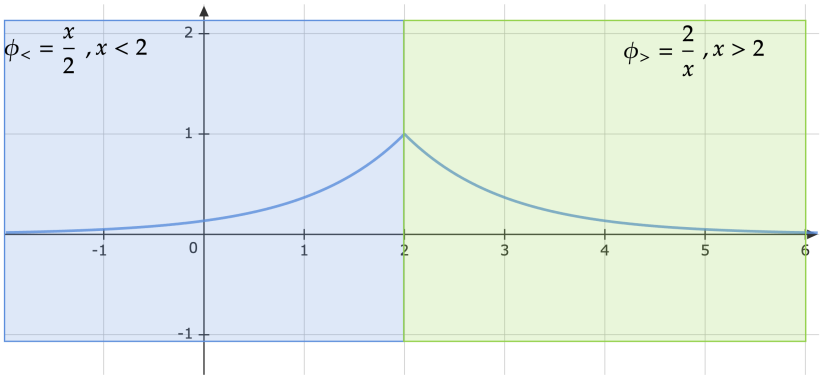


Figure 5.6: An example of the functions less than (blue area) and greater than (green area) with exponential decay, where the query value is equal to 2.

Exponential descending order: The exponential variants of bound-based conditions are shown in Equation 5.5, where $\Phi_>$ and $\Phi_<$ are a constraint version of the equality score in Equation 5.3.

$$\Phi_>(v_{x_1}, v_i) =: \begin{cases} exp(-|v_{x_1} - v_i|) & v_{x_1} > v_i \\ 0 & else \end{cases} \tag{5.5}$$
$$\Phi_<(v_{x_1}, v_i) =: \begin{cases} exp(-|v_{x_1} - v_i|) & v_{x_1} < v_i \\ 0 & else \end{cases}$$

As illustrated in Figure 5.6, altering the functions from ratio to exponential results in a more balanced scoring between the two conditions. Similar to the equality function, values in close proximity to the quantity, adhering to the numerical condition bounds, are assigned higher rankings than those with more substantial differences.

In our experiments, we noticed that the slope in which the score convergences to zero in the exponential setting is too steep for values with a large magnitude, leading to a sub-optimal ranking. Conversely, in ratio functions, the proximity is unaffected by the magnitude of the value. Take the query "Family house less than 100k dollar" and two sentence quantities, q_1 = (90,000, "dollar") and q_2 = (80,000, "dollar") into account. The ratios $\frac{v_1}{v_{x_1}} = \frac{90{,}000}{100{,}000} = 0.9$ and $\frac{v_2}{v_{x_1}} = \frac{80{,}000}{100{,}000} = 0.8$ provide a meaningful comparison of proximity, while the exponential decay for both quantities converges to zero. While this behavior is desirable for the equality function, it becomes less suitable for the bound-based conditions, where a broader range of values should be taken into account. As a result, for all experiments throughout this chapter, the ratio functions are employed.

Ascending order: The sorting order can be reversed from descending to ascending by subtracting each corresponding ratio or exponential decay score from 1. As an example, we denoted the ratio variants in ascending order in Equation 5.6, the exponential version is computed in a similar manner.

$$\Phi_{>}(v_{x_1}, v_i) =: \begin{cases} 1 - v_{x_1}/v_i & v_{x_1} > v_i \\ 0 & else \end{cases}$$
$$\Phi_{<}(v_{x_1}, v_i) =: \begin{cases} 1 - v_i/v_{x_1} & v_{x_1} < v_i \\ 0 & else \end{cases} \tag{5.6}$$

For the example query of "iPhone XS $<$ 1,500 dollars", quantities q_1 = (1,490, "dollar") and q_2 = (800, "dollar") would have another sorting based on Equation 5.6. In this case, the scores are reversed, $1 - \frac{v_1}{v_{x_1}} = 1 - \frac{1{,}490}{1{,}500}$ and $1 - \frac{v_2}{v_{x_1}} = 1 - \frac{800}{1{,}500}$, and q_2 is ranked higher. In this sorting, the quantity farthest from the query value is deemed the most relevant.
In a real-world scenario, both settings (ascending and descending) could be valid depending on the intention of the user. For the example query of "iPhone XS $<$ 1,500 dollars", the user could be looking for the cheapest "iPhone XS" or an "iPhone XS" slightly below "1,500 dollars".

5.1.2.3 Range Ranking Function

In general, when one defines a valid range, any value that falls within the query range of v_{x_1} and v_{x_2} is of relevance. In this section, we discuss different variations to enforce various sortings based on proximity to (1) lower bound, (2) upper bound, and (3) average value. These are all propositions of various ways the results can be displayed to the user and are best chosen by the user themselves during search and not pre-defined by a system architecture.

Average of bounds, descending order: In this case, we assign a higher weight to values closer to the *average of the lower and upper bounds* (v_{x_1} and v_{x_2}), where the weight exponentially decays as the values diverge from the mean. $\Phi_{[]}$ for average of bounds is shown in Equation 5.7, which assigns a score based on the exponential difference between $v_{avg} = \frac{v_{x_1}+v_{x_2}}{2}$ (average of lower and upper bound) and v_i (sentence value). This results in sorting the values based on their proximity to the average, with values closer to the average ranked higher than those further away. While there are no strict bounds, values significantly outside the range receive extremely low scores due to the exponential decay function.

$$\Phi_{[]}((v_{x_1}, v_{x_2}), v_i) =: exp(-|v_{avg} - v_i|) \tag{5.7}$$

If the user requires strict bounds, Equation 5.7 can be modified to assign a score of zero to any value outside the specified range, as shown in Equation 5.8. In this case, only values within the range are sorted based on the proximity to the average of the lower and upper bound.

$$\Phi_{[]}((v_{x_1}, v_{x_2}), v_i) =: \begin{cases} exp(-|v_{avg} - v_i|) & v_{x_1} < v_i < v_{x_2} \\ 0 & else \end{cases} \tag{5.8}$$

Average of bounds, no sorting: Alternatively, to remove the current sorting based on the average value, one could simply apply hard filtering and assign a score of 1 for values within the range and an exponential decay (or zero score) for values outside, as shown in Equation 5.9. In this case, all values within the range are equally relevant and their distance to the average is irrelevant. Yet, their relevance decreases exponentially as they fall out of the valid range.

$$\Phi_{[]}((v_{x_1}, v_{x_2}), v_i) =: \begin{cases} 1 & v_{x_1} < v_i < v_{x_2} \\ exp(-|v_{avg} - v_i|) & else \end{cases} \tag{5.9}$$

Lower/upper bounds, descending order: To sort based on proximity to the lower or upper bound, we can use a combination of $\Phi_{>}$ and $\Phi_{<}$. For ranking values closer to the lower bound higher, we use the ratio from $\Phi_{>}$ as shown in Equation 5.10. This method is particularly suitable for scenarios like users searching for car prices within a range, where they typically prefer the lowest possible price within their specified range.

$$\Phi_{[]}((v_{x_1}, v_{x_2}), v_i) =: \begin{cases} v_{x_1}/v_i & v_{x_1} < v_i < v_{x_2} \\ 0 & else \end{cases} \tag{5.10}$$

For the exponential decay variation, we change the ratio to the exponential decay of the lower bound, as shown in Equation 5.11.

$$\Phi_{[]}((v_{x_1}, v_{x_2}), v_i) =: \begin{cases} exp(-|v_{x_1} - v_i|) & v_{x_1} < v_i < v_{x_2} \\ 0 & else \end{cases}. \tag{5.11}$$

The same strategy applies to sorting based on the upper bound, and equations are similar to Equations 5.10 and 5.11. The only difference is that instead of the lower bound value, the upper bound value is used.

Ascending order: This type of sorting does not make sense for the ranges condition and is therefore not discussed here.

5.1.3 Term Ranking

The disjoint quantity-centric ranking requires a separate model for term-based ranking. This part can come from any lexical or semantic ranker, requiring only normalized scores. Nonetheless, retrieval systems rarely care about the normalization of relevance scores between 0 and 1. Not only it is computationally expensive, but also it does not affect the final ranking. In the following, we discuss possible ways to normalize the term ranking scores of various neural and lexical models and integrate them with the quantity ranking. For a lexical model, we use BM25, discussed in Section 2.1.3, and for neural models we choose ColBERT and SPLADE, discussed in Sections 2.2.6.1 and 2.2.7.1 respectively, as two representatives of dense and sparse neural rankers.

With recent trends in the retrieval community and the focus on neural models, one might wonder about the usefulness of pursuing a lexical ranker altogether. The main issue with lexical rankers stems from their use of one-hot encoding for words. These encodings are incapable of representing semantics and meaning in their representation, where all words have the same distance from one another. Synonyms, plurals, and miss-spellings are not captured by these models [5], requiring custom made synonym and spelling variation lists and pre and post-processing of inputs. Ambiguous and context-dependent words are mapped to the same representation, which causes additional problems, e.g., "Apple Inc." and "Apple Pie" are not the same.

We argue that despite these shortcomings of the lexical system, there are still several use cases where lexical models are a better fit. An example from the DPR paper (Karpukhin et al., 2020) reflects this difference the best, given the question "Who is the bad guy in lord of the Rings?", DPR (a neural model) can successfully return the relevant span "Sala Baker is best known for portraying the villain Sauron in the Lord of the Rings trilogy" by capturing the semantic similarity between "villain " and "bad guy". However, when it comes to exact matching, lexical models are more effective, i.e., keywords or named entities. Another example from the DPR paper (Karpukhin et al., 2020) reflects such a case, given the query "Who plays Thoros of Myr in Game of Thrones?" looking for an exact lexical match for *Thoros of Myr* is critical in finding the correct passage. Dense models often return noisy data when an exact matching is required, where the exact search terms are not guaranteed to be in the top-k. This case becomes more extreme when entities or compositions are rare and not part of the training set.

All things considered, widely used open-source search engines such as Lucene [6] still use lexical matches as their main search functionality. The unsupervised nature of lexical models, minimum computational cost, and their domain-independent compatibility make them an attractive option for industrial purposes, in contrast to neural models, which often require expensive GPU power for efficient training and inference. To this end, we provide a way to integrate the quantity score with both neu-

[5] Although stemming and lemmatization are proposed to map the plurals and different verb tenses to the same representation, the nature of stem can cause further problems, e.g., the company name "Withings" is reduced to "With", which is often removed as a stop word.

[6] https://lucene.apache.org/ Last accessed: 02.05.2024

ral and lexical models, such that the user can decide based on their specific use cases which model to employ.

5.1.3.1 Lexical Model

We present a lexical variant of our approach with a probabilistic term-based model, specifically BM25 (Robertson and Zaragoza, 2009). This is the most prominent approach in many IR systems and thus is easy to integrate into existing frameworks. BM25 is described in detail in Section 2.1.3, therefore the details are excluded from this section.

BM25 relies on a term-based index and standard pre-processing steps before the index creation. These steps include removing punctuation and stop words, converting text to lowercase, and stemming. An inverted index entry is then created for each unique token, linking to a posting list of sentences containing that token.

BM25 uses the created term-based index and statistics of the corpus to compute the relevance of query terms to sentences in the corpus. The general calculation of the BM25 score from Section 2.1.3 is adjusted to sentence structure in Equation 5.12. For this study, we consider the commonly used Okapi BM25 (Robertson et al., 1994; Robertson and Zaragoza, 2009) variant.

$$\text{BM25}(s_i, T_x) = \sum_{t \in s \wedge T_x} \frac{s_t(1+k_1)}{s_t + k_1((1-b) + \frac{b.dl}{avgdl})} log \frac{|S| - |S_t| + 0.5}{|S_t| + 0.5} \cdot \frac{T_{x_t}(1+k_2)}{T_{x_t} + k_2} \tag{5.12}$$

The BM25 score is unbounded and, depending on the query and statistics of the query terms in a given corpus, has various minimum and maximum values. In order for the quantity and term-based scores to be comparable, they must share the same scale. Therefore, we need to normalize the BM25 score. For this purpose, for each individual sentence, the score ($\text{BM25}(s_i|T_x)$), is divided by the maximum BM25 score for the query terms across all sentences ($\frac{\text{BM25}(s_i,T_x)}{\max_{s_i \in S}(\text{BM25}(s_i,T_x))}$).

To obtain the quantity-centric ranking, the quantity score, calculated using the heuristic functions, is added to the normalized BM25 score. This final score is shown in Equation 5.13 as *QBM25 (Quantity-aware BM25)*. The parameter α controls the impact of the quantity ranking. When set to zero, the formula falls back to term-based ranking. For higher precision, we restrict the addition of a quantity score to sentences where all the search terms are present, denoted by the indicator function $\mathbb{1}_{T_x}(s_i)$. For instance, for the query "Apple Inc. financial reports with a yearly profit of more than $2 million", returning information about a different company would defeat the intended purpose. If the quantity score is added to all sentences regardless of the lexical matches, then a sentence such as "Amazon has a revenue of $2.1 million" would get an undeserving high score due to the quantity proximity, when the textual information does not match the query constraints.

$$\text{QBM25}(s_i, c, x) := \frac{\text{BM25}(s, T_x)}{\max_{s_i \in S}(\text{BM25}(s_i, T_x))} + \alpha \mathbb{1}_{T_x}(s_i) \text{QScore}(s_i, c, x) \tag{5.13}$$

5.1.3.2 Neural Dense Model

As described in Section 2.2.6, neural dense models are divided into two main families, term-based representations and document-level representations. In document-level models, a single embedding encodes the semantics of a document. In term-based models, token embedding and their interactions are considered for more fine-grained and powerful retrieval systems. Our quantity ranking approach is integrable with both term and document-level models. Here, we describe the integration with the term-level model of ColBERT, described in detail in Section 2.2.6.1. This choice is due to the same model being used for joint quantity ranking, where token-level interactions are crucial. The same paradigm is applicable to document-level models.

Recall the similarity computation between the document and a query for the ColBERT model from Section 2.2.6.1, shown in Equation 5.14. The document representation is swapped with sentence representations to fit the constraints of our model, and we replaced the generic τ function with BERT encoders. ColBERT utilizes two BERT (Devlin et al., 2019a) encoders for query and document (sentence), where each encoder outputs a list of token embeddings.

$$\text{ColBERT}(s_i, T_x) = \sum_{i \in |\text{BERT}(T_x)|} \max_{j \in |\text{BERT}(s_i)|} \text{BERT}(T_x)_i \cdot \text{BERT}(s_i)_j \tag{5.14}$$

The ColBERT score is computed with the MaxSim operator, where the maximum cosine similarity of each token embedding in a sentence with vectors in a query embedding is computed and combined via summation. This value, similar to the BM25 score, is unbounded and requires normalization. However, in the context of neural models, calculating the similarity for the entire corpus to identify the maximum value for each query is impractical. During ranking, the ColBERT model does not compute this score for the entire corpus. Instead, ColBERT leverages the pruning-friendly nature of the MaxSim operator to take advantage of the approximate nearest neighbor search, detailed in Section 2.2.5.2, to return the top-k most relevant candidate sentences (S_k). We employ a similar approach and compute the maximum score based on the top-k candidate sentences (S_k), to normalize the score between 0 and 1, $\frac{\text{ColBERT}(s_i, T_x)}{\max_{s_i \in S_k}(\text{ColBERT}(s_i, T_x))}$. Subsequently, the quantity score is incorporated exclusively into the top-k candidates, acting as a second-stage re-ranker, to re-order the candidate set according to numerical proximity.

The final score for each sentence is the weighted summation of the ColBERT score and the quantity score, where α controls the impact of quantity ranking. We refer to this score as *Quanity-aware ColBERT (QColBERT)*, as shown in Equation 5.15.

$$\text{QColBERT}(s_i, c, x) := \frac{\text{ColBERT}(s_i, T_x)}{\max_{s_i \in S_k}(\text{ColBERT}(s_i, T_x))} + \alpha QScore(s_i, c, x) \tag{5.15}$$

Note that in comparison to QBM25, the quantity score does not affect the entire ranking and is only applied to the subset retrieved by ColBERT. Therefore, if a relevant sentence is not among the

top-k the quantity score does not have an impact. To this end, we also present our disjoint quantity ranking with a sparse neural model, where the quantity score can be directly integrated into the final ranking of all the sentences in the corpus.

5.1.3.3 Neural Sparse Model

Another family of neural retrieval systems is the sparse retrieval models, described in Section 2.2.7. Like the lexical models, many of these systems end up with a similar inverted index structure to check for lexical overlap on representations of documents and queries, which is often expanded with new terms using a neural architecture.
The SPLADE model, detailed in Section 2.2.7.1, extends the document and query terms using the BERT embeddings and stores the expanded representations in an inverted index. In this index, instead of term frequencies, the importance weights are computed by the BERT model. This characteristic of the SPLADE model makes it well-suited for integration with our proposed quantity ranking, as the quantity score can be directly incorporated into the final ranking.
For each sentence, the SPLADE model computes the sentence embeddings with BERT and passes it through a ReLU non-linearity and log function to produce a sparse vector over the entire vocabulary. The values in this vector are predominantly close to zero, except for the vocabulary terms that are deemed important by the BERT model. The query is expanded in the same way, and the similarity between the query and a sentence is based on the sparse dot product of this representation, as shown in Equation 5.16.

$$\text{SPLADE}(s_i, T_x) := \log(1 + \text{ReLU}(\text{BERT}((s_i))) \cdot \log(1 + \text{ReLU}(\text{BERT}((T_x))) \tag{5.16}$$

This dot product is efficiently computed for all sentences inside a given corpus by multiplying the non-zero elements in a query and sentence vector, which are stored in an index structure.
To obtain a score between 0 and 1, the SPLADE score is normalized based on the maximum score for a given query, $\frac{\text{SPLADE}(s_i,T_x)}{\max_{s_i \in S}(\text{SPLADE}(s_i,T_x))}$. The final ranking for *Quantity-aware SPLADE (QSPLADE)* is shown in Equation 5.17 and is achieved by the summation of the normalized SPLADE score with the quantity score for each sentence. For higher precision, we restrict the addition of a quantity score to sentences where there is a match between the expanded query terms and sentences, denoted by the indicator function $\mathbb{1}_{\text{SPLADE}(T_x)}(s_i)$.

$$\text{QSPLADE}(s_i, c, x) := \frac{\text{SPLADE}(s_i, T_x)}{\max_{s_i \in S}(\text{SPLADE}(s_i, T_x))} \cdot \alpha \mathbb{1}_{\text{SPLADE}(T_x)}(s_i) \text{QScore}(s_i, c, x) \tag{5.17}$$

Note that in contrast to the ColBERT model, the quantity score impacts the ranking of all the sentences with a term match to the query and not solely the top-k candidates.

A framework that disjointly ranks the term-base similarity and quantity proximity has many advantages. It can be combined with a variety of term-based ranking methods, and does not require training data or fine-tuning expensive neural architectures. In the following section, we discuss the shortcomings of the disjoint approach and motivate the joint ranking of quantities and terms.

5.2 Joint Ranking of Quantities and Terms

The main limitation of the disjoint ranking is the independence assumption between the relevance of quantities and terms. Assume the example query "iPhone XR below €200". Since the term-based ranking is computed independently from the quantity score, both of the following sentences might receive an inappropriately high score.

- *The price of iPhone XR has reached €236.50 over the past month, whereas the older version of Samsung A14 is as low as €132.00.* In this case, the numerical condition is satisfied but for a quantity related to a different concept than the one in the query, while the relevant concept does not satisfy the numerical condition in the query.
- *Older versions of iPhone, including iPhone XR have dropped in price with refurbished versions of iPhone 8 as low as €152.94 on Amazon.* Here, the relevant concept has no quantity associated with it in the text, and the relevant quantity is related to an irrelevant concept.

Both of these cases occur due to the lack of a correct association between a concept and a quantity. We call this problem the *quantity-concept mismatch*.
One possible solution is to utilize the full output of CQE during the creation of the quantity index and to consider the concepts as well as units. More precisely, the inverted index for the quantity can be extended to a triplet of *(concept, unit, value)*, pointing to sentences containing all three. This index can be used instead of the quantity index described before, for both term and quantity ranking. However, this solution has a few shortcomings. Concepts are multi-token spans and indexing them as single tokens results in a significant increase in index size, thus, reducing the performance. Additionally, this index structure inherits all the typical issues of traditional lexical-based systems, such as vocabulary mismatches in concepts and errors from stemming and lemmatization. Furthermore, any errors in concept detection will propagate to the retrieval phase, and because we limit the search context to detected concepts, recall might drop dramatically. A better solution is to seek a way to remove the quantity and term separation and score sentences based on the entire context of a query.

Definition 5.3. *(Joint Quantity-centric Ranking).*
Given a quantity-centric query (as defined in Definition 5.1), the relevance, $r(s_i|x)$, of a sentence s_i to a query x in a joint quantity-centric ranking is defined as

$$r(s_i|x) \sim sim(\tau(x), \tau(s_i)) = sim_c(\tau(T_x, q_x), \tau(T_i, Q_i)).$$

The similarity function sim_c is dependent on the query condition c, and ranks the textual content as well as the quantity proximity. τ is a generic function that maps the query and sentences in a given corpus to their representations.

Regarding the choice of τ, lexical models are not ideal. These models assume independence between all tokens in a document (or sentence) and query, making them unsuitable for modeling inter-token dependencies. In contrast, neural retrieval models based on the transformer architecture account for the inter-dependencies of tokens within the entire context. These models learn patterns and relationships as long as examples are provided to them in the training set. As mentioned in the motivation of this chapter, the current retrieval benchmark lacks quantity-centric queries and thus suitable training data for this task is not available. Therefore, it is unknown if the lack of quantity understanding in this model is due to a lack of task-specific training data or if the token representation and model architecture of current systems prevents them from learning numerical comparisons [7]. We aim to answer this question by designing a data generation paradigm for quantity-centric ranking, using the CQE framework. We automatically generate queries from the concepts and quantities in the corpus and create positive and negative sentences through value and unit permutation.
The designed framework aims to provide positive and negative examples of cases where dense models make the most mistakes. After interacting with the dense models and investigating the error cases, two main sources of errors were identified:

- Dense models are unable to perform value comparisons given numerical conditions. For the example query "iPhone X with a price lower than 880€", the models ignore the less than condition, where sentences containing both an "iPhone X with 990€" or with "iPhone X with 500€" are ranked unjustifiably high.

- Traditional training paradigms based on the topic similarity between query and document on dense models backfire in the case of units. For the example query "iPhone X with a price lower than 880€", if the corpus contains many instances with dollar or other currencies, sentences containing those currencies can be ranked higher than sentences containing euro as a unit, due to the semantic similarity of currency units.

As a result data generation is focused on providing contrastive examples for such cases by means of data augmentation. In the following subsections, we go over the data and query generation process. Neural models fine-tuned on this task-specific dataset exhibit a better understanding of numerical conditions and quantities in text.

[7] In Section 3.3.1, we mentioned several related work that point out the shortcoming of current word embedding and transformer models in dealing with numbers in text. While keeping those shortcomings in mind, Wallace et al. (2019) also point out that most neural embeddings work decently on numeracy tasks for values in the range of training set.

5.2.1 Joint Quantity Ranking Training Paradigm

The data generation is specifically designed to enhance (1) value comparisons and (2) understanding of unit surface forms, by generating positive and negative sentences through augmentation. Data augmentation is a prominent method for generating additional, synthetic data and is widely used in computer vision. In recent years, data augmentation has also been used in NLP tasks, e.g., in back translation (Sennrich et al., 2016a). In Section 3.4, GENBERT and ENCORE models (Geva et al., 2020; Wang et al., 2024) were mentioned that utilize templates to generate pre-training data for a question answering system capable of numerical reasoning on text and table. GENBERT shows that by pre-training the BERT model on automatically generated synthetic data with numerical templates, the model's numerical reasoning is enhanced without resorting to specialized architectures. In this section, we follow the same approach by adding a fine-tuning step with automatically generated data to improve the quantity understanding in current retrieval models without the change in architecture.

Following the terminology of Geva et al. (2020), we also view our approach as data augmentation. However, given that we are perturbing values and units in a sentence, one might alternatively label this approach as data perturbation. Alternatively, due to enriching the sentences with quantity information context enrichment or data expansion might come to mind. However, for the remainder of this chapter, we stick to the notion of data augmentation.

The pre-requisite for this data generation paradigm is a dataset rich in quantitative information and an extractor capable of identifying values, units, and related concepts. The CQE framework from Chapter 4 provides us with the necessary extraction capabilities, and it is utilized for data generation. The same paradigm can be applied with any other extractor, provided that the essential elements are successfully extracted.

The data generation pipeline has three main stages:

1. *Quantity Extraction:* A quantity extractor is used to enrich the corpus with quantity information by extracting values, units, and related concepts from each sentence.

2. *Query generation:* By utilizing the quantity information related to concepts in the corpus, and a set of templates, a set of quantity-centric queries is generated.

3. *Sample generation with value and unit permutation:* For each query, additional positive and hard negative samples are generated by permuting the values and units in sentences.

In the following, we describe each step in more detail.

5.2.2 Quantity Extraction

The first step is enriching the corpus with quantity information by means of a quantity extractor. The documents are split into sentences and each sentence is fed independently to CQE to extract

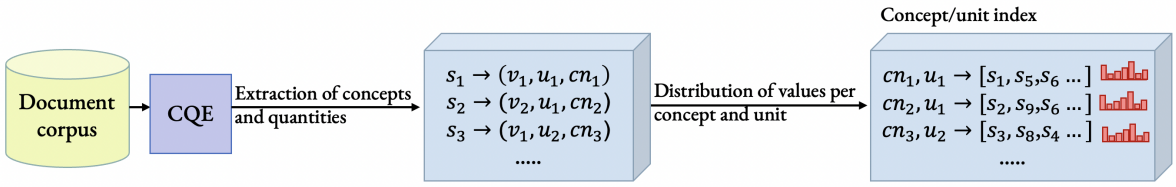


Figure 5.7: Overview of the quantity tagging step and creation of concept/unit index structure, where the red arrows signify the peak in the distribution and the yellow box shows the values around the mean.

values, units, and related concepts. Based on the extracted quantities, the corpus is transformed into an index-like structure based on the related concepts and units. We refer to this structure as *concept/unit index*. The index keys are concept/unit pairs that refer to the same quantity. Each key points to a list of values associated with the concept/unit and a list of the sentences in which they appear. This list of values represents the distribution of values for a given concept and unit. For example, an entry in the index from the dataset used in the evaluation is shown below.

```
{("cannabis company", "cent per share"):
{"values":[1.4, 17.0, 17.0, 22.0, 26.0, 35.0, 84.0],
"sentences":["The cannabis company says the loss amounted to 0.9 of a cent per share for
the quarter ended May 31 compared with a loss of 4 million dollar or 1.4 cents per share a
year earlier .",
"The cannabis company says its loss amounted to 17 cents per diluted share for the quarter
ended Jan. 31 .",...]}}
```

Note that repetitions of values for the same concept/unit pair are stored as duplicates to maintain the frequency distribution, e.g., the value "17.0" is stored twice as it occurs in two distinct sentences. The steps in the creation of the concept/unit index are illustrated in Figure 5.7. The corpus is processed with CQE to extract values, units, and concepts from each sentence, where sentences sharing the same unit and concept are grouped into a list, along with values represented as a distribution. We take advantage of this index in the next steps to generate queries and training samples.

5.2.3 Query Generation

The queries for the training set are created by processing the concept/unit index. For each concept/unit pair, three queries, one for each condition, equal, less than, and greater than, are created with the template:

```
query = {concept} {numerical_condition} {unit_before}{value}{unit_after}.
```

The variables enclosed in the brackets are populated during query generation. For more clarity, we also provide an algorithmic view of the query generation in Algorithm 1 and explain in the following how each placeholder is filled.

A quantity-centric query has three main elements, a *quantity* (value/unit pair), a *concept* that the quantity is referring to, and a *numerical condition* imposed upon the value of the quantity. The concept/unit index derived from the quantity extraction stage provides a list of typical concepts and

units in the corpus and the range of values commonly associated with them. Utilizing this distribution of values as a reference, we select values to satisfy three numerical conditions (equal, less than, and greater than). Queries for ranges are more complex and we leave this matter for future work. For each concept/unit pair, three queries, one for each condition, are created with the given template.

Algorithm 1 Query Generation

function GENERATE_QUERY($conc, unit, condition$)
  $v \leftarrow$ get_query_values($conc\_unit\_dict, condition$) ▷ select a value
  $u\_b, u\_a \leftarrow$ get_unit_surfaceform($unit$) ▷ select unit surface form
  $c \leftarrow$ get_condition_surfaceforms($condition$) ▷ select wording of the condition
  $query \leftarrow conc + c + u\_b + v + u\_a$
  return query
end function

---

conc_unit_dict $\leftarrow$ concept/unit index
conc_expand_dict $\leftarrow$ concept expansions
for $(concept, unit)$ in conc_unit_dict **do**: ▷ for all concept/unit pairs
  for $conc$ in [$concept$, conc_expand_dict[$concept$]] **do**: ▷ find expansion
    for $condition$ in $(equal, greater, less)$ **do**: ▷ for all conditions
      GENEARATE_QUERY($conc, unit, condition$)
    end for
  end for
end for

Unit: The concept/unit index contains normalized units. Consequently, if we rely solely on the surface form of the unit within the index, our training data would only include a single surface form per unit. To account for variability in the representation of units, we utilize a dictionary of unit surfaces provided by CQE to augment the query value with different surface forms. To this end, a surface form of the query unit is chosen randomly from a dictionary of units, e.g., "€" is a surface of the unit "euro". Unit surface forms include symbols and abbreviations, which can appear before or after a quantity value. Hence, two placeholders are used in the query template for units, where one is left empty in each query. `unit_before` and `unit_after` account for symbols appearing before, e.g., "€" and abbreviations after a value, e.g., "EUR".

Value: Each entry in the concept/unit index points to the sentences and list of values that co-occur within a given corpus. To generate samples, we need both positive and negative instances per query. By choosing query concepts and units from the index, we guarantee the presence of supporting sentences containing their co-occurrence. Yet, to find positive sentences, the value and the numerical

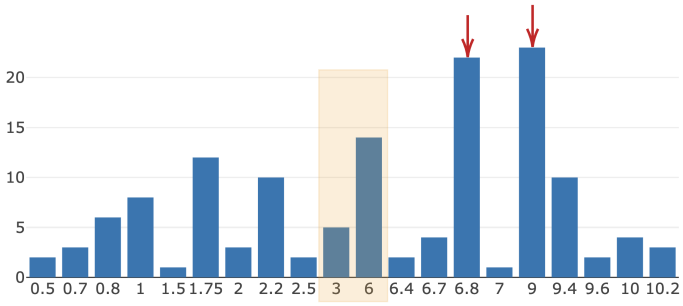


Figure 5.8: An example value distribution from the concept/unit index, the peaks of the distribution marked with red arrows are suitable candidates for queries with equal conditions. The values close to the average, marked with a yellow box are suitable candidates for the bound-based conditions, less than and greater than.

condition should also be satisfied. Therefore, selecting the query value becomes crucial to ensure the presence of supporting sentences in the index. We propose the following strategy:

- *Equal query:* For the equal query, the challenge is to find enough positive samples, since there is an abundance of not equal values in each distribution. The values with the highest frequency, representing the peaks of the distribution, are determined based on the number of sentences containing them. Therefore, for each concept and unit pair, values for equal queries are selected from the most frequent values. This process is illustrated in Figure 5.8, where the values at the identified peaks (indicated by red arrows) are selected for the query with equal condition. In this manner, we make sure that there are enough positive samples in the corpus for the data augmentation.

- *Less than and greater than queries:* These numerical conditions correspond to bounds and cover a range of values. For such queries, we avoid infrequent values towards the tail of the distribution, to avoid too few positive or negative samples. Optimal candidates for satisfying these bounds are values close to the average, ensuring a broader coverage of values within the specified limits. Selecting infrequent values towards the tail of the distribution may result in inadequate supporting sentences. In Figure 5.8, for instance, values close to the average (highlighted in a yellow box) are chosen for the queries with conditions based on bounds.

To avoid systemic bias by focusing on the most frequent values, we generate a second set of queries by picking the values at random.
To account for variability in representation, large values that have multiple written surface forms are randomly replaced with their written form in the query template. The written form may take the shape of a composite of numbers and postfixes, such as "10 million," or include commas for convenient digit separation, for instance, e.g., “10,000,000”. This variability occurs only for values above 1000. Decimal numbers, on the other hand, possess a singular representation, e.g., “10000” can be represented as “10,000” or “10k” or “10 thousand” but “10.2” has only a single representation.

Numerical Condition: A numerical condition is a phrase in natural language indicating a bound on a quantity. For this purpose, a surface-form dictionary is established for the three numerical conditions, and the corresponding placeholder is populated with surface forms randomly selected from this dictionary. The dictionary is shown in Table 5.1, containing multiple surface forms for each condition. This list is not comprehensive but provides enough variety for query generation.

Table 5.1: Numerical conditions used for query generation and their surface forms.

Condition	Surface forms
Equal	exactly, exact, equals, equals to, for, with, of, at, =
Greater than	greater than, more than, above, larger than, over, higher than, exceed, exceeding, >
Less than	smaller than, below, less than, fewer than, no more than, beneath, <

Concept: Queries generated by taking a concept from the concept/unit index are guaranteed to have positive samples in the corpus. However, the concept extraction in CQE identifies multi-word spans in a sentence as concepts, and utilizing them directly for query generation overlooks the nuances of semantic queries. For example, in the sentence "Walt Disney Co. launched Disney+ in November, charging $6.99 a month for access to its trove of films and shows .", "Disney+" is the extracted concept for the quantity "(6.99, dollars per month)". A valid query in this case would be "Disney+ equal to $6.99 a month". Nonetheless, "Disney+" falls under the umbrella of streaming platforms, encompassing entities like "Netflix" and similar media services. Hence, if a user searches for "streaming platform equal to $6.99 a month", the same sentence should be retrieved as relevant. Relying exclusively on queries with exact matching keywords in sentences poses a risk of biasing the trained models toward keyword search and away from semantic search. To avoid such a case, we add *concept expansion* to the query generation pipeline, where a large language model like GPT-3 is used to generate synonyms or synsets for a given concept. The expansions are generated before the actual query generation process, from the list of extracted concepts in the concept/unit index, and stored in a dictionary for efficient use. In conjunction with exact matching keywords, these expansions generate semantic queries. For each expanded concept, new value and unit surface forms are sampled to generate a semantic query for each numerical condition.
For example, consider the concept/unit pair "(Apple Inc Gross Profit Margin, percentage)" that points to the value distribution "[26.70, 26.70, 26.70, 26.70, 22.46, 38.51, 38.51, 44.52, 45.17, 45.88, 45.88, 45.88]". The following lists six random queries that can be generated:

1. Apple Inc Gross Profit Margin = 26.70%
2. tech company GM exactly 45.88 percent
3. Apple Inc Gross Profit above 38.51 pct.
4. tech company GM over 22.46%

5. Apple Inc. Gross Profit below 44.52 percentage

6. tech company GM under 45.88 percent

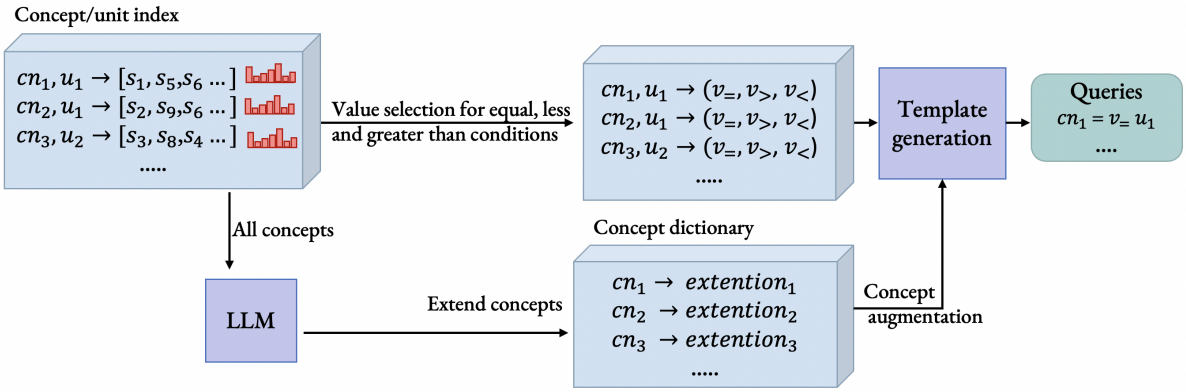


Figure 5.9: Overview of the query generation pipeline, using the output of the quantity tagging stage and a large language model for concept expansion.

The complete query generation pipeline is depicted in Figure 5.9. The concept/unit index from the previous stage is used on one hand to select values and units for numerical conditions. On the other hand, a large language model is used to expand concepts and avoid lexical queries. The template generation block combines all the outputs to formulate three queries for each unit and concept pair. For expanded concepts, new values are sampled from the value distribution and a new set of queries is formulated for each numerical condition.

5.2.4 Sample Generation with Value and Unit Permutation

Semantic retrieval systems consider an entire context to find a fuzzy relevance to the query at hand. Often, this aligns well with the user's expectations. For instance, when searching for a "dark color evening dress", any dress that can be worn as an evening gown and has a dark color would be suitable. But as soon as the user becomes more specific like "blue evening dress", the embedding space could also bring a similar color like "teal" into the search result. Depending on the user's flexibility regarding the dress color, this behavior may or may not be desirable. Such hard constraints are challenging for neural models. Quantity-centric queries compose hard constraints on values and units, where the fuzzy matching of context might do more harm than good. For instance, when searching for a "car with more than 320 hp", if the results contain a car with "360 brake horsepower" instead of horsepower the result is irrelevant. Both horsepower and brake horsepower are used in similar contexts but correspond to different properties. Horsepower measures the power generated by the engine, while brake horsepower measures how much of the power produced by the engine is sent to the wheels which makes the car accelerate. Another common problem is with currencies. Given that monetary values often appear in similar contexts, it becomes challenging for a model to differentiate between various currency units. Hence, it is crucial for a model to grasp the significance of units in

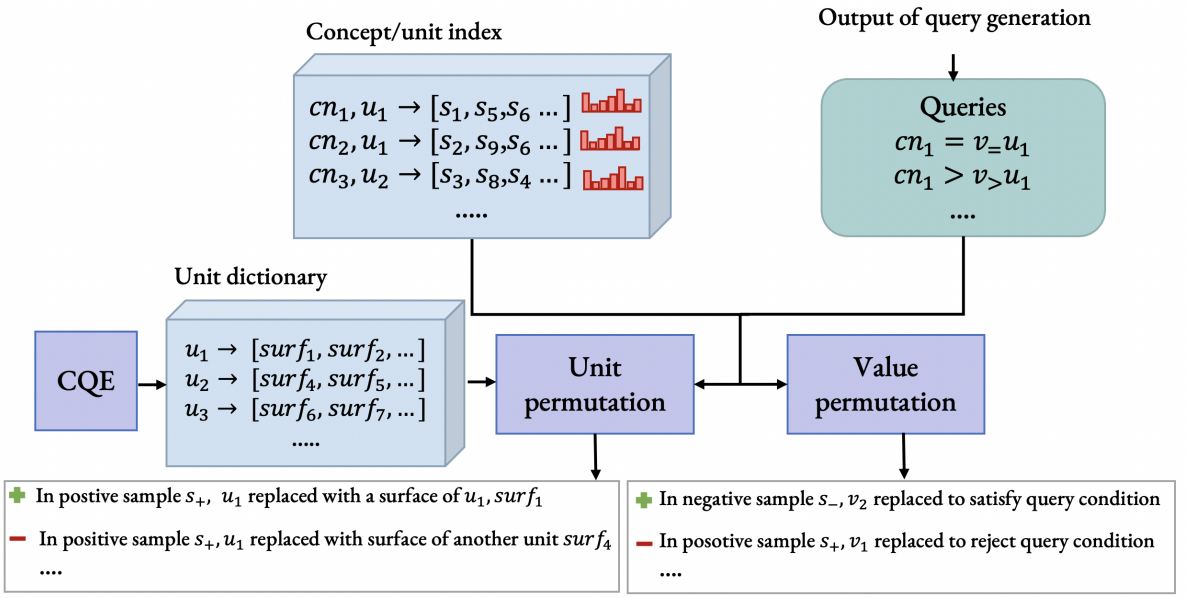


Figure 5.10: Overview of the sample generation using value and unit permutation.

alignment with the query context.
The second essential point is the constraints on numerical values. The model must possess the ability to compare the magnitude and scale of the value in a query with the values in sentences. When searching for a "car with more than 320 hp", a sentence containing "310 horsepower" is irrelevant under the greater than condition.

Both of these criteria need to be represented in the training data, with contrastive examples enabling the model to learn the correct ranking strategy. An outline of our sample generation pipeline for positive and negative examples is shown in Figure 5.10. The inputs of this stage are the generated queries and the concept/unit index. For each quantity-centric query, beyond the positive and negative samples obtained from the dataset, additional samples are generated through unit and value permutation, where the unit surface forms are chosen from a dictionary provided by CQE.
The initial positive and negative samples are sourced from the list of sentences associated with each concept and unit pair in the concept/unit index. Applying the query condition to the list of values allows for generating a list of sentences that either conform to or violate the condition. The original positive and negative samples are then chosen at random from such a list.
The same list is utilized as seed samples for data augmentation. Unit and value permutation are employed to generate augmented positive and hard negative samples. Hard negative are positive samples, where the unit or value is perturbed to violate the query condition.
The steps are presented in Algorithm 2 for a more structured understanding. Each sampling mechanism is encapsulated within a distinct function, and the final training samples are the union of all generation mechanisms. In the following, we describe each step and function in more detail.

Algorithm 2 Sample Generation

s_- ← set of sentences violating the query condition
s_+ ← set of sentences conforming the query condition
n ← sample size
function ORIGINAL_SAMPLING(s_+, s_-, n)
    return sample(s_+,n), sample(s_-,n) ▷ sample n sentences
end function

$unit$ ← query unit
n ← sample size
s_+ ← set of sentences conforming the query condition
function UNIT_PERMUTATION(s_+, n, $unit$)
    s_{u+} ← replace_same_unit_surface(s_+, $unit$)
    s_{u-} ← replace_other_unit_surface(s_+, $unit$)
    return sample(s_{u+},n), sample(s_{u-},n)
end function

$condition$ ← query condition
v ← query value
$values$ ← list of values for query unit and concept
n ← sample size
s_- ← set of sentences violating the query condition
s_+ ← set of sentences conforming the query condition
function VALUE_PERMUTATION(s_+, s_-, n, $values$, v, $condition$)
    s_{u+} ← replace_with_positive_value(s_-, v)
    s_{u-} ← replace_with_negative_value(s_+, $value$, $condition$)
    return sample(s_{v+},n), sample(s_{v-},n)
end function

---

conc_unit_dict ← concept/unit index
queries ← list of queries
n ← number of samples
for $(concept, unit, condition, value)$ in queries **do**: ▷ for each query
    s ← conc_unit_dict[$(concept, unit)$]
    s_+, s_- ← filter_based_on_condition(s, $condition$, $value$)
    s_{o+}, s_{o-} ← **ORIGINAL_SAMPLING**(s_+, s_-, n)
    s_{u+}, s_{u-} ← **UNIT_PERMUTATION**$(s_+, n, unit)$
    s_{v+}, s_{v-} ← **VALUE_PERMUTATION**$(s_+, s_-, n, values, v, condition)$
    $s_{f+} = s_{o+} \cup s_{u+} \cup s_{v+}, s_{f-} = s_{o-} \cup s_{u-} \cup s_{v-}$
end for

Look-up: Given a query containing a quadruple *(concept, unit, condition, value)*, a lookup is conducted in the concept/unit index to retrieve the sentences and the distribution of values associated with the concept and unit specified in the query.

Positive and Negative Sentences List: The obtained sentences are divided into positive (s_+) and negative (s_-) lists, based on the numerical condition. s_+ contains sentences where the values satisfy the query condition, and s_- contains sentences violating the query condition. For instance, given the equal condition, if the sentence value (v_s) is equal to the query value ($v_{i=}$), it will be part of s_+. Conversely, if v_s violates the condition ($v_s \neq v_{i=}$), it is included in s_-.

Original sampling: With sample size n, sentences are randomly selected from s_+ as positive samples (s_{o+}) and from s_- as negative samples (s_{o-}). These samples are added to the training set as original positive and negative examples. If the number of available sentences in the positive list or negative lists is smaller than the sample size, a *downsampling* procedure is implemented. When $|s_+| < n$ or $|s_-| < n$, the sample size is reduced to the smallest number of available samples.

Unit permutation sampling: This method generates positive and negative samples to cover diverse unit surface forms using CQE's unit dictionary. Positive samples contain various surface forms of the unit in the query (u_i), while negative samples include surface forms of other units in the same family as the query unit, creating negatives. The unit permutation is applied to only the positive sentence list, with n samples chosen from s_+.

- Positive samples (s_{u+}) are formulated by substituting the unit in positive sentences (s_+) with other surface forms of the same unit in query u_i.

- Negative samples (s_{u-}) are created by replacing the unit in positive sentences (s_+) with a surface form of a unit different from the query unit (u_i) but belonging to the same family. The unit families are defined by the CQE framework, where there is a grouping of the units based on the property they measure. For example, "pace", "meter", "yard", "inch" and "foot" all belong to the same family of "length". Sampling the surface form from the same family ensures a fine distinction between unit types, even in similar contexts.

Permutations of the noun-based units are rather limited, where the positive samples are left unchanged and negative samples are generated by random replacement with another noun-based unit from the corpus. For instance, in the sentence "I ate 3 apples", the unit "apple" remains the same for positive instances, and for negative permutations, we consider all the other noun-based units in the corpus as possible permutations.

Value permutation sampling: This permutation emphasizes the importance of the value comparison and numerical conditions, highlighting that sentence relevance depends on whether the sentence value satisfies the query condition or not.

- Positive samples (s_{v+}) are formulated by permuting the values in negative sentences (s_-), maintaining the correct concept and unit and adjusting the value to satisfy the condition.

- Negative samples (s_{v-}) are generated by permuting the values in positive sentences (s_+). In this case, the unit and concept align with the query but the value is permuted to invalidate the query condition.

The replacement values are sampled from the values in the concept/unit index, mirroring the underlying distribution of the relevant quantity. It is crucial that the permuted values obey the original value distribution of the corpus. The properties of concepts are often limited to a specific range, e.g., the value "10000" is unreasonable for the percentage rate of unemployment. Moreover, certain values are on a discrete scale with limited options, e.g., "RAM of a laptop" is limited to distinct values such as 4, 8, and 16. Assigning a random number outside this range, like 10, would be unlikely. Therefore, for the synthetic data to obey the rule of the real-world dataset and reflect the distribution of different properties, the permuted values are chosen from the values observed in the corpus.

Aggregate: The final set of positive (s_{f+}) and negative (s_{f-}) samples for each query is the union of all samples generated from the original sampling, value and unit permutation, $s_{f+} = s_{o+} \cup s_{u+} \cup s_{v+}$ and $s_{f-} = s_{o-} \cup s_{u-} \cup s_{v-}$.

The models reported in the evaluation use a combination of original sampling with unit permutation and concept expansion on the query. Value permutation did not show stable performance gains, which we attribute to the difficulty of numerical representations in dense models. A more elaborate discussion on this topic is given in Section 5.4.6.

With both the query and sample generation, a training set of augmented data is generated that contains queries for the three numerical conditions as well as contrastive examples. This training set can be paired with any neural retrieval architecture to enhance the quantity understanding with additional fine-tuning. Since the generated data focuses only on queries with numerical conditions, this set alone is not enough to acquire a full-fledged retrieval model. It is best to use the generated data for an extra fine-tuning step using an already trained checkpoint on general domain queries. The models discussed for the joint quantity ranking in the evaluation are trained in this manner.

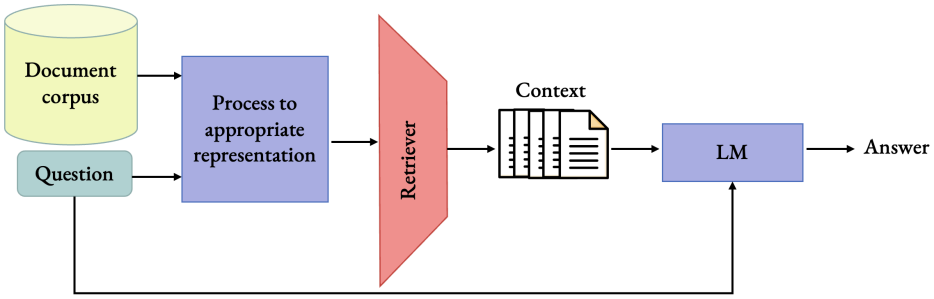


Figure 5.11: Overview of a vanilla RAG pipeline, where the context passages or documents are retrieved using a retriever and passed to a language model along with the question.

5.3 Integration into RAG Pipeline

The evolution of large language models in recent years has touched the field of NLP, including retrieval systems and QA. Within this landscape, one prominent application is integrating external knowledge sources with the help of retrieval models to a language model, this method is referred to as *Retrieval Augmented Generation (RAG)* (Guu et al., 2020; Lewis et al., 2020).
In this section, we consider RAG with quantity-centric ranking as a downstream application of our ranking models and discuss their applicability to answer quantity-centric questions. We begin with a brief overview of current research in this domain and then describe our integration.

5.3.1 What is RAG?

Large language models, trained on millions of documents, capture a vast amount of knowledge about the world, enabling them to answer questions without relying on external data sources. However, this knowledge is confined to the training data. As a result, they lack the ability to address questions about events occurring after their training or in domains not covered by their training corpus. There is also a struggle to memorize knowledge that is rarely mentioned (Kandpal et al., 2023) and their memory cannot be easily updated or erased. Furthermore, since model parameters offer no insight into predictions, identifying issues when incorrect answers are generated or hallucinations occur can be challenging (Marcus, 2020).
One solution to these problems is to couple the language model with an external memory, as done in a RAG pipeline. In RAG, the language model is contextualized by relevant passages retrieved from an external corpus. These passages, along with the original question, are then provided to the language model for answer generation. By providing access to an external knowledge base, the language model does not have to rely solely on the knowledge encoded in its parameter but can answer questions from an unknown domain or recent data. Furthermore, by grounding the answer on the retrieved passages, the origin of the response becomes clearer. This reduces instances of hallucination and provides attribution by referencing the specific document from which the answer was derived. A general RAG pipeline is depicted in Figure 5.11. The retriever can take the form of either

a traditional index structure or a dense neural model. RAG systems can be categorized into three types based on which part of the pipeline is updated during task-specific training: *Frozen RAG*, *RAG with contextualized retriever*, and *RAG with both contextualized retriever and generator*. Below, we elaborate on these approaches.

5.3.1.1 Frozen RAG

In the frozen RAG (Borgeaud et al., 2022) approach, no training is performed. Both the language model and the retriever have been trained independently beforehand and are now coupled together in a pipeline similar to Figure 5.11. This method heavily depends on the language model's capability for in-context learning (Brown et al., 2020), enabling it to adjust and generate responses according to the given context. The accuracy of a response depends on two key factors: firstly, the retriever's ability to find the correct context, and secondly, the language model's adaptability in leveraging this context to produce the correct answer.

Outside academic settings, frozen RAG is a paradigm used most often, and a variety of methods have been proposed to enhance the quality. The efforts mainly focus on methods to enrich the retrieved chunks of data with additional context or fuse the output of different retrieval models, often referred to as *advanced frozen RAG*. The main advances in this domain come from the developer community and frameworks such as LlamaIndex [8] and LangChain. [9] In the following, we point to a few approaches in this domain:

Context enrichment: Retrieval units depend on how the data is chunked and play an important role both for retrieval and generation. Splitting must be done in a way that ensures enough context for the language model to reason upon but it is specific and small enough for efficient search. One approach is to adjust the chunk size differently for retrieval and generation tasks. Retrieval can be performed on smaller chunks, with additional surrounding context later added for the language model to reason. Two common methods for incorporating additional context include *sentence window retrieval* and *parent document retriever* (Gao et al., 2023b).

In sentence window retrieval, once the most relevant sentences are identified, the context is expanded by including k sentences before and after each retrieved sentence. This extended context is then sent to the language model for reasoning and answer generation.

Parent document retriever also known as *auto-merging retriever* follows a similar concept. In this approach, documents are split into smaller child chunks, which correspond to larger parent chunks, e.g., paragraphs. If more than n chunks within the top k retrieved chunks are associated with the same parent, the context is replaced with the larger parent chunk.

[8] https://www.llamaindex.ai/ (last accessed 02.05.2024)
[9] https://www.langchain.com/ (last accessed 02.05.2024)

Hierarchical indices: With a large collection, there is the need to efficiently search, find relevant information, and synthesize it into a single answer. A hierarchical index essentially consists of two indices: one for summaries and another for document chunks. The retrieval is performed in two steps, by first composing a candidate set of the relevant documents by finding relevant summaries and then searching inside this relevant group.

Hypothetical questions and HyDE: Gao et al. (2023a) propose an idea to fix hallucination through purposeful hallucinations. Given a query, the language model is asked to generate hypothetical answers. The embedding of the hypothetical answers is then used to find relevant chunks in the corpus. The reverse of this is also possible, where the language model is asked to generate a question for each chunk and embed these questions in vectors. The chunk vectors are replaced with generated question vectors in the index and utilized for retrieval. After retrieval, the original chunks are used as context for the language model. These methods improve the quality of retrieval since there is a higher semantic similarity between query and hypothetical questions compared to an actual chunk.

Fusion retrieval or hybrid search: This method combines the classical retrieval and neural retrieval models using reciprocal rank fusion (Rackauckas, 2024). To properly combine the score of the two models Reciprocal Rank Fusion algorithm (Cormack et al., 2009) is used. Hybrid or fusion search usually provides better retrieval results as two complementary search algorithms are combined, taking into account both semantic similarity and keyword matching between the query and documents.

Re-ranking and filtering: Similar to two-stage retrieval systems, a re-ranker can be used to refine the results (Nogueira and Cho, 2019). The choice of re-ranker can vary from another large language model with zero-short ranking capabilities or a trained cross-encoder (Déjean et al., 2024).

Query transformations: These are a family of techniques using language models as a reasoning engine to modify user input in order to improve retrieval quality (Zhou et al., 2023; Trivedi et al., 2023). For example, if the query is complex, the language model can decompose it into several subqueries, e.g., for the question "Is the unemployment rate of the UK higher than the US?", it is unlikely to find a direct comparison in some text. It is better to decompose this question into two sub-queries to find the unemployment rate in the UK and the US and then compare them. Other forms of query transformations are *Step-back prompting* (Zheng et al., 2023), which uses a language model to generate a more general query, and *Query re-writing*, which uses a language model to reformulate the initial query.

For our experiments, we focus on the frozen RAG setting but provide a brief introduction to other RAG types as well.

5.3.1.2 Contextualizing the Retriever

Another family of methods aims to adapt the retriever for the generator without the need to alter the weights of the generator itself. One example of this is the *RePlug* model by Shi et al. (2023), where the retriever is trained using the output of a frozen language model as supervision. During training, the generator language model is used as a ranking function to measure how much each document could improve the language model's perplexity. For each retrieved document, the perplexity of the ground truth output answer given the input context is computed using the generator, and the retriever is trained by minimizing the KL divergence between the perplexity of the document and the retrieval score for that document. The advantage of this approach is that it works well for any type of generator and retrieval method and is not restricted to a specific model type.
Another approach in this domain is *In-Context RALM* (Ram et al., 2023), which is essentially a frozen RAG with BM25 and a trained ra-ranker. In this case, the BM25 model and generator are left untouched but back-propagation is performed on the weights of the re-ranker to increase the probability of retrieving the document that produces the correct answer.

5.3.1.3 Contextualizing the Retriever and the Generator

In this scenario, both the retriever and generator are optimized and the entire architecture is contextualized. One of the first methods in this domain was proposed by Lewis et al. (2020), where both the generator and the retriever are updated during training. Two methods were proposed based on the timing of the retrieval process: the RAG sequence model and the RAG token model. In the RAG token model, a new context is retrieved before generating each token. In contrast, the RAG sequence model requires only a single retrieval to generate an entire sequence. They showed that optimizing the whole pipeline works substantially better than any frozen variant. Later, methods such as *FLARE* (Jiang et al., 2023) advise to not prescribe the time of retrieval but to let the language model learn when to rely on retriever knowledge for a generation.
Another example contextualizing both retriever and generator is the *KNN-LM* model (Khandelwal et al., 2020), which extends a pre-trained language model by linearly interpolating it with k-nearest neighbors from a corpus. The idea is to interpolate between the parametric memory of the language model and the non-parametric memory of the corpus to re-weight the language model generation probability. Given an input text, the model generates the output distribution over the next words and the retriever finds its k-nearest neighbors. Based on the distance of the input and retrieved context and the frequency of tokens, a probability distribution over the tokens in the retrieved context is created, which is interpolated with the language model's next-word generation probability.
In the mentioned methods the update would only take effect for the query encoder. The *REALM* model (Guu et al., 2020), which was published before the hype of RAG models, was the first that introduced a way to update the document encoder. REALM is a BERT-based language model augmented with a retrieval component. For pre-training, the masked language modeling is altered to

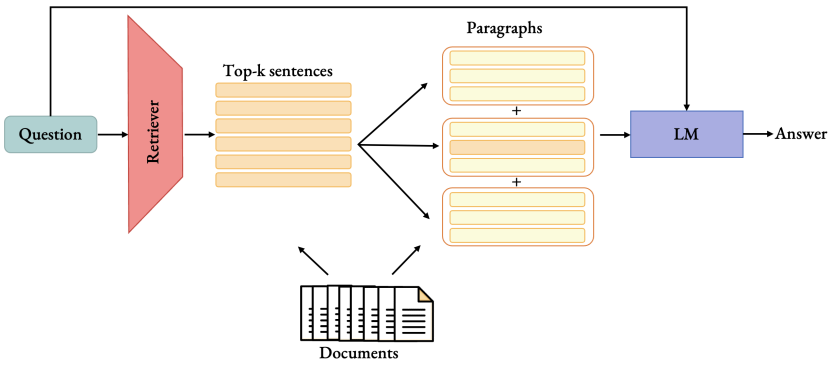


Figure 5.12: Overview of the RAG pipeline for our quantity-aware model involves retrieving context sentences using a retriever, contextualizing them with paragraphs, and then passing them along with the input question to a language model to generate an answer.

predict salient spans such as named entities or dates, using the retrieved evidence from the corpus. Since updating the document encoder requires re-indexing the entire document collection, the authors perform asynchronous updates after certain batches of training. This involves re-embedding and re-indexing all documents every few hundred training steps.

5.3.2 Quantity Ranking with Augmented Generation

A retriever's ability to find the relevant context is an essential part of a RAG system. As the work of Liu et al. (2023) highlights even the order in which the context is presented to the language model has an impact on whether the correct information is utilized during answer generation. Liu et al. (2023) emphasize the beginning and the end of the context is most important and the information in the middle is somewhat lost during generation in frozen RAG settings.

Finding the correct context for questions that contain numerical conditions depends on the retrieval system's ability to deal with quantity-centric queries. For example, for a quantity-centric question of "Which states of the united states have unemployment rate less than 6%?", the ranking model should retrieve text chunks for "unemployment rate" that satisfy the numerical condition. If the retrieved chunks have topic relevance (information about the unemployment rate in various states) but incorrect quantities (unemployment rate does satisfy the condition of less than 6%), then the generation model is unlikely to produce the correct response.

To investigate the impact of a quantity-centric retrieval in a RAG setup, we propose to replace a general domain retriever with a quantity-aware one for answering quantity-centric questions (the red block in Figure 5.11). We focus mainly on a frozen RAG set-up with a hierarchical index for context enrichment in the form of a child-parent chunk retrieval. In this case, the retrieval units remain as sentences but each retrieved sentence is enriched by the content of the parent paragraph and a paragraph before and after. For retrieving the surrounding context, a hierarchical index is created as shown in Figure 5.12, where the retrieval is performed on the child index and the parent index is only utilized for context enrichment in the form of surrounding paragraphs.

Since we focus on frozen RAG, neither the retriever nor the generator are fine-tuned. We leave such endeavors for future work.

5.4 Evaluation of Quantity-centric Ranking

In this section, we evaluate our models against various baselines on ranking performance, and downstream performance (in a RAG setting).
The absence of task-specific models leads us to assess our quantity-aware models only against general domain lexical and neural models. Moreover, the lack of benchmark data for this task is addressed by introducing two English resources named FinQuant and MedQuant. To the best of our knowledge, these are the first quantity-centric benchmarks for quantity-aware IR.
For the evaluation of quantity-centric ranking, we start with an overview of the baselines employed for comparison in Section 5.4.1, followed by a detailed presentation of the datasets developed for this task in Section 5.4.2. We provide a comprehensive evaluation of our joint and disjoint models on FinQuant and MedQuant datasets. To this end, we first introduce the evaluation metrics commonly used in IR in Section 5.4.4 and motivate the metrics we chose for comparison. The evaluation in Section 5.4.5 reports the comparison of the general model performance of FinQuant and MedQuant datasets, as well as subsets of the data. The subsets are designed to compare models on handling semantic versus lexical queries, and across different numerical conditions.
In the description of the joint quantity-aware ranking, we provided various data generation strategies. In Section 5.4.6, we provide an ablation study on these strategies for choosing the best combination. Moreover, to investigate whether task-specific fine-tuning has shifted the ranking strategy of the neural models to pay attention to quantity tokens (values and units in the text), in Section 5.4.7 we look at the impact of masking tokens related to quantities on the overall ranking performance.
In order to look at the effect of quantity-aware ranking in the context of RAG systems, in Section 5.4.8, we introduce the relevant metrics for assessing the quality of a generated answer. Then, the factual consistency and correctness of answers for a set of 42 questions quantity-centric questions is assessed and reported.
The code for all the models discussed here and the FinQuant and MedQuant datasets are available in our repository: `https://github.com/satya77/QuantityAwareRankers`.

5.4.1 Baselines

Our baselines are divided into lexical and neural models.

Lexical models: The lexical rankers include a BM25 and a BM25$_{filter}$ variant. For the lexical models, the corpus and the test queries undergo pre-processing steps to normalize the unit surface forms in the text, e.g., all instances of “km/h”, “kilometre per hour” or “kilometer an hour” is replaced

with "kilometer per hour", such that the difference in unit representation does not hinder the baselines from finding the correct result. As a result, the units in the corpus as well as the unit in the query text are normalized, such that the inverted index in the lexical models can easily map the correct unit. BM25$_{filter}$ has a separate numerical index for quantity values, where the distinct values in the corpus are the keys of the index and point to sentences containing them. This index is utilized to eliminate the results of BM25 where the query condition is not met. This method resembles numerical indices from databases, focusing on filtering rather than ranking.

Neural models: Our neural baselines include the trained checkpoints of SPLADE and ColBERT as well as Cohere$_{v3}$ [10]. SPLADE and ColBERT checkpoints are trained on general-purpose retrieval datasets, which lack quantity-centric queries. For both models, we utilized the best-performing checkpoints provided by the authors.
Cohere$_{v3}$ embeddings are provided through the Cohere API and are included to show that even industry-level models trained on extensive data still lack quantity understanding.

5.4.2 Datasets

In this section, we describe the creation of FinQuant and MedQuant datasets. For dataset creation, similar to the generation of synthetic data, we first construct the concept/unit index. Test queries are manually formulated using the concept/unit index for each dataset, covering both lexical and semantic queries as well as all three numerical conditions (equal, less than, greater than). Statistics for various query types are presented in Table 5.2. Each numerical condition is equally represented, with 140 queries per condition in FinQuant and 70 queries per condition in MedQuant. Keyword-based queries are those with lexical matches in the corpus and are divided into two categories of *seen* and *unseen*. Semantic queries constitute a smaller portion due to annotation challenges and are divided into *exapnsion* and *w/o surface form*. These categories are explained in more detail in the upcoming section.

Table 5.2: Statistics of the different types of queries in FinQuant and MedQuant datasets.

		FinQuant	MedQuant
	Total queries	420	210
	Sentence in corpus	306,291	153,252
Per condition	Equal queries ("=")	140	70
	Greater than queries(">")	140	70
	Less than queries ("<")	140	70
Keyword-based queries	keywords seen in training (*seen*)	150	60
	Keywords not seen in training (*unseen*)	150	60
Semantic queries	Keyword synonyms or supersets (*expansion*)	75	30
	Keywords not present in corpus (*w/o surface form*)	45	60

[10] https://cohere.com/embeddings (last accessed: 02.05.2024)

FinQuant is created from a set of news articles in the categories of economics, science, sports, and technology, collected between 2018 and 2022. The FinQuant corpus contains over 300k sentences from 473,375 news articles. MedQuant is smaller, containing over 150k sentences from 375,580 medical documents of the TREC Medical Records (Voorhees, 2013) on Clinical Trails.

5.4.2.1 Annotation of Test Set

The test sets are annotated by the author of the thesis and a student assistant. Inter-annotator agreement is computed for each dataset on a subset of 20 samples per dataset. The Cohen's Kappa coefficient (Cohen, 1960) is 0.83 and 0.88 for FinQuant and MedQuant, respectively, showing a high agreement between the annotators.
Datasets were split into sentences and processed to eliminate boilerplate HTML or headers. All sentences containing quantities were incorporated into the collection. The entire test data is manually created and tagged. Here, we describe the query formulation and annotation tasks.

Query formulation: The concept/unit index and the value distributions from the data generation pipeline were used to formulate queries. Annotators were tasked with creating quantity-centric queries based on the concept/unit index. They scanned the entire index for possible synonyms related to each concept, formulated queries accordingly, and maintained a list of these synonyms. For example, if an annotator selects "Microsoft Surface Earbuds" with the unit "pound sterling", they would then scan the concept/unit index for other relevant concepts associated with "pound sterling". This process would identify related terms such as "Earbuds" and "Microsoft headphones".
The value distribution and sentence list from all the synonyms of a concept were consolidated into one and presented to the annotator. The annotator was then instructed to choose three values from this distribution for equal, less than, and greater than queries, in such a way that supporting sentences for the query are present. In the final stage, the annotator formulated a query in natural text using various surface forms of units and numerical conditions, e.g., "Microsoft Surface Earbuds lower than 179 pounds". The annotators had access to the dictionary of surface forms for units and numerical conditions to help query formulation and avoid repetitive queries.

Candiate list generation: For each query, a list of possibly relevant sentences was generated using the concept/unit index. To achieve such a list, all sentences related to the concept and the concepts synonyms (from the consolidated list in the previous step) were filtered based on the query value and the numerical condition. The filtering process is automated to reduce the annotation effort, presenting the annotator with a refined list of sentences that match the query value and condition for a final review. For instance, for the query "Microsoft Surface Earbuds lower than 179 pounds", all sentences that contain 'Microsoft Surface Earbuds", "Earbuds" and "Microsoft headphones" with unit "pound sterling" and the value in the sentence is less than "179" are added to the candidate list.

Since CQE is used for the creation of the concept/unit index, its performance directly affects the data quality. While CQE is adept at handling financial data, extractions on clinical data were noisy, impacting performance comparisons later on. However, we find it important to report results on both datasets, making the reader aware of the lesser quality of MedQuant. In both datasets, there is no guarantee that the candidate list is comprehensive and covers all relevant instances.

Annotation: An annotation guideline was devised to instruct the annotators to set up the annotation pipeline correctly and for consistent annotation of ambiguous cases. The guideline is published alongside the dataset in our repository. Annotators were presented with the list of candidate sentences (from the previous step) for each query and were tasked to mark the relevant sentences. The marked sentences were used as ground truth for the subsequent evaluation.

5.4.3 Implementation Details

The code is implemented in Python 3.10 and PyTroch 1.13. The general sentence splitting and text cleaning were performed with SpaCy 3.9. [11] As mentioned before we use the CQE library [12] for quantity extraction. Evaluation and metrics were computed with the help of the `pytrec_eval` library (Van Gysel and de Rijke, 2018). [13] In the following, we discuss the implementation details for each model separately.

BM25 models: We use the Okapi BM25 package [14] for all BM25 variants. The QBM25 and BM25$_{filter}$ are variations of Okapi BM25 designed to include a quantity index for ranking and filtering and were implemented by forking the same package and making alterations to the code. The parameters of BM25 were tuned to each of our datasets separately, as presented in Table 5.3. The latency values (reported later on) are computed with plug-ins for an Opensearch instance [15] (described in more detail in Section 5.5) on a desktop computer with 16GB of RAM. In comparison to the dense models, the lexical models do not require specific hardware architectures.

Table 5.3: Hyperparameters of BM25-based models on the benchmark datasets.

	FinQuant	MedQuant
BM25	b = 0.5, k1 = 0.5,	b = 0.5, k1 = 0.5
BM25$_{filter}$	b = 0.75, k1 = 1.5	b = 0.75, k1 = 1.5
QBM25	b = 0.5, k1 = 0.5	b = 0.5, k1 = 0.75

[11] https://spacy.io/ (last accessed: 02.05.2024)
[12] https://github.com/satya77/CQE (last accessed: 02.05.2024)
[13] https://pypi.org/project/pytrec-eval/(last accessed: 02.05.2024)
[14] https://pypi.org/project/rank-bm25/ (last accessed: 02.05.2024)
[15] https://opensearch.org/ (last accessed: 02.05.2024)

Cohere baseline: We used the Cohere API [16] for Cohere$_{v3}$ embeddings. The Cohere API offers two types of embeddings. *Query embeddings* were used to encode the queries and the *document embeddings* to encode the collection.

ColBERT models: Khattab and Zaharia (2020) supplied the trained checkpoint for the base ColBERT model. For fine-tuning on augmented data, the model was initialized with this base checkpoint [17]. The checkpoint without further training was employed for the evaluation of both ColBERT and QColBERT. ColBERT$_{ft}$ was fine-tunned using the training script from the official repository. [18] The code in the repository was forked and modified to establish an endpoint for QColBERT incorporating a quantity index. We did not perform extensive hyperparameter tuning except for the learning rate and mainly used the parameters advised by the authors for both FinQuant and MedQuant datasets. We fine-tuned ColBERT$_{ft}$ for 2 epochs, with a batch size of 256 and a learning rate of 1e-05 on a server with four A-100 GPUs and 40GB of memory. The evaluation and benchmarking for latency were performed on the same server, utilizing all four GPUs.

SPLADE models: SPLADE$_{ft}$ was also fine-tuned using the training script by the authors. [19] The pre-trained checkpoint was acquired from Hugging Face [20] and utilized for both the SPLADE model and QSPLADE. Scripts from the official repository were adjusted to add a quantity index for QSPLADE. Similar to ColBERT, we conducted limited hyperparameter tuning, mainly focusing on the learning rate. We fine-tuned SPLADE$_{ft}$ for 2 epochs using a batch size of 240, a learning rate of 2e-5, and a weight decay of 0.01. The fine-tuning was conducted on a server with four A-100 GPUs and 40GB of memory. The evaluation and benchmarking for latency were performed on the same server, utilizing all four GPUs.

For all disjoint rankers, QBM25, QColBERT, and QSPLADE, the quantity impact parameter of α is set to 1, such that the impact of term and quantity ranking is equal.

Generated data: Based on the combination of augmentation methods, the size of training data would vary. In all cases, we saved a small sample of 1000 queries for validation. There were 40,732 and 20,376 concept and unit pairs considered for query generation in FinQuant and MedQuant, respectively. If concept expansion is applied, these numbers would double to account for queries on expanded concepts. We set the sample size n to 2, meaning that for each augmentation strategy, two instances are generated in addition to positive and negative samples from the data without augmentation. As a result, for each query, we have n sample from the data and based on augmentation

[16] https://cohere.com/ (last accessed: 02.05.2024)
[17] The checkpoint to the model is not publicly available but if one emails the authors the access is usually granted
[18] https://github.com/stanford-futuredata/ColBERT (last accessed: 02.05.2024)
[19] https://github.com/naver/splade (last accessed: 02.05.2024)
[20] https://huggingface.co/naver/splade-cocondenser-ensembledistil (last accessed: 02.05.2024)

methods, additional samples are added for concept expansion and unit and value permutation, a total of $3n$ per query. In the case of concept expansion and both value and unit permutation and 40,732 queries with $n = 2$, we have a collection of $40,732 \times 3 \times 2 = 244,392$ sentences. Note that this value might be slightly lower in practice due to the down-sampling described in Section 5.2.4.

Concept expansion: For concept expansion, we use the OpenAI API [21] and employ the `text-davinci-003` model with few-shot learning. We set the temperature to 1 to encourage creative responses (other parameters were left to the default setting). Since the concepts come from the two different domains of finance and medicine, the few-shot examples vary accordingly. Below we specify the two prompts used for concept expansion, the result is stored in a *concept expansion dictionary* and utilized during query generation. The place-holder `{concept}` is replaced with a concept to be expanded from the concept/unit index.

For the finance domain:

```
Complete with the words superset or synonym, but do not reuse the exact same words, the
word "Super Set" should not be in the response and response should have at least two words:

S&P 500 = stock market index
Audi = car
Oil prices = petroleum prices
unemployment rate = unemployment percentage
iPhone sales = phone sales
Netflix shares = stock shares
President Trump = president
iPhone 11= iPhone
Hong Kong = city
stake PEXA = Property Exchange Australia shares

{concept} =
```

For the medical domain:

```
Complete with the words superset or synonym, but do not reuse the exact same words, the
word "Super Set" should not be in the response and response should have at least two words:

ophthalmic solution = eye medication
Control group = treatment group
irinotecan hydrochloride = chemotherapy drug
monoclonal antibody = substitute antibodies
MRI scans = magnetic resonance imaging
influenza H1N1 vaccine = flu vaccine
HAI antibody response = Influenza-specific antibody response

{concept} =
```

[21] https://openai.com/ (last accessed: 02.05.2024)

5.4.3.1 Semantic and Lexical Queries

The queries are categorized into four types: *seen*, *unseen*, *expansion*, and *w/o surface form*. The lexical queries consist of *seen* and *unseen* subsets. For such cases, during query formulation, the annotators picked concepts from the concept/unit index without the change in surface form. For example, if the concept "NASDAQ index" is chosen, the exact terminology is used for query generation. The concepts from the *unseen* category, were removed from the index for data generation and training of the joint neural models. Therefore, it contains lexical queries that were not seen during training. For example, "YouTube channel" is a concept in the *unseen* subset, which means all instances of "YouTube channel" were removed from the concept/unit index before data generation.
Semantic queries consist of *expansion* and *w/o surface form* subsets and were slightly harder to formulate, thereby, fewer instances of them are present in the data. For *expansion* queries, a concept from the lexical set was chosen to expand to one of its supersets or synonyms. For example, "YouTube channel" is a concept from the *unseen* category that is expanded to "social media channel" as a semantic concept. These expansions were used to formulate queries that did not have a lexical match in the database and often included a superset of many concepts. In the case of "social media channels", other social media channels like "Facebook" or "Twitter" are also considered as relevant. The *W/o surface form* subset is created similarly to the *expansion* queries, except that the concepts are not chosen from the *seen* and *unseen* categories.

5.4.4 Evaluation Metrics

In this section, we introduce the metrics we use to measure the effectiveness of IR systems. Basically, the metrics are divided into *order-aware* and *order-unaware* metrics, indicating if the order of the results has an impact on the final score (Manning et al., 2008). Order-unaware metrics include, *Perision@K (P@K)*, *Recall@K (R@K)*, and *F1@K*. These are set-based measures that can be computed on an unordered set of documents and are not limited to a retrieval setting. All three measures were already covered in Chapter 4, so here we do not go into their details again. The only difference is the addition of @K, where the results are computed up to a certain rank. In the evaluation that follows, we will present two of the order-unaware metrics, specifically P@K and R@K.
The popular order-aware metrics are *Mean Reciprocal Rank (MRR@K)*, *Mean Average Precision (MAP@K)*, and *Normalized Discounted Cumulative Gain (NDCG@K)*. In the following, we briefly describe the intuition behind each metric. In the evaluation that follows, we present two of the order-aware metrics in terms of MRR@K and NDCG@K.

5.4.4.1 Mean Average Precision (MAP)

Average precision (AP) is the average of precision scores calculated at the positions where relevant documents are retrieved. Basically, it involves examining the ranks at which relevant documents appear, computing precision at each of these ranks, and then taking the mean across these precision

values, as shown in Equation 5.18. P@k represents the precision calculated for the top-k retrieved documents, and rel_k indicates the relevance of the item at position k. AP operates with binary relevance, so this value is set to 1 for relevant items and 0 for non-relevant items.

$$\text{AP} := \frac{\Sigma_k^n \text{P@k} \times rel_k}{n} \tag{5.18}$$

MAP is the arithmetic mean of the average precision values for a set of X test queries (Manning et al., 2008; Liu and Özsu, 2018), as shown in Equation 5.19.

$$\text{MAP} := \frac{1}{|X|} \sum_{x=1}^{|X|} \text{AP}_x \tag{5.19}$$

In Mean Average Precision, the *mean* refers to the average calculated across multiple queries and *average* specifically denotes the average of precisions at individual ranks within a single query. When we use MAP@K, it implies that the evaluation cuts off the results at rank K. Similar to precision, this metric is biased towards the top of the ranking. Typically, an arithmetic average is employed for aggregating results across multiple queries, although in certain cases, the *Geometric Mean Average Precision (GMAP)* may also be utilized (Robertson, 2006).

5.4.4.2 Mean Reciprocal Rank (MRR)

MRR is another measure that is particularly useful in scenarios where the goal is to measure the quality of the top-ranked results. For a given query, the *Reciprocal Rank (RR)* is the inverse of the rank at which the first relevant item is found. If the first relevant item is at rank r then RR=$\frac{1}{r}$. For instance, RR is 1 if a relevant document was retrieved at rank 1, if not it is 0.5 if a relevant document was retrieved at rank 2, and so on. MRR is the average of Reciprocal Ranks across all queries, as shown in Equation 5.20 for queries in the set X.

$$\text{MRR} := \frac{1}{|X|} \sum_{x=1}^{|X|} \text{RR}_x \tag{5.20}$$

MRR@K is a variant that focuses exclusively on the top-k results. MRR operates under the assumption that one scans through the ranking until one encounters a relevant document and stops one's search. If the first relevant document is at rank r, then the precision value at this rank is also $1/r$, which makes RR and AP equal. For this reason, MRR is equivalent to MAP in cases where each query has precisely one relevant document (Liu and Özsu, 2018).

5.4.4.3 Normalized Discounted Cumulative Gain (NDCG)

One of the most popular metrics for IR systems is NDCG@K. It assesses the quality of the ranking by considering both the relevance and the position of the items. NDCG (Järvelin and Kekäläinen,

2002) is designed specifically for cases where there are graded degrees of relevance. The relevance scores are computed based on a utility function rel, where each document in the ranking has a utility associated with it. The optimal ranking would then prioritize documents with the highest utilities at the top. In the context of binary relevance, utilities are set to zero and one, where a document is either relevant (with rel equal to 1) or non-relevant (with rel equal to 0). In this case, the utility function resembles that of MAP.
We start with *Cumulative Gain (CG)*, as shown in Equation 5.21. rel_k indicates a fine-grained scoring, typically between 0 (least relevant) and 4 (most relevant).

$$\text{CG@K} := \sum_{k=1}^{K} rel_k \tag{5.21}$$

CG lacks order-awareness. To introduce order into ranking, we leverage Discounted Cumulative Gain (DCG) and add a log penalty value in the denominator. DCG is calculated by summing the relevance scores of the items at each position, discounted by the logarithm of their positions as shown in Equation 5.22.

$$\text{DCG@K} := \sum_{k=1}^{K} \frac{rel_k}{\log_2(1+k)} \tag{5.22}$$

DCG values are unbounded and their range is dependent on rel values. To address this, the NDCG is introduced, utilizing *Ideal DCG (IDCG)* for normalization purposes (Manning et al., 2008). IDCG represents the highest possible DCG for a given set of items and their true relevance scores. It is obtained by sorting the items based on their true relevance scores and computing the DCG value. Then, NDCG is calculated by normalizing DCG by its ideal counterpart as shown in Equation 5.23.

$$\text{NDCG@K} := \frac{\text{DCG@K}}{\text{IDCG@K}} \tag{5.23}$$

A higher NDCG indicates a better quality ranking, taking both relevance and position into account.

5.4.4.4 Which Metrics to Use?

There is a lot of controversy regarding the correct metric and evaluation strategies for IR systems (Fuhr, 2017; Sakai, 2020). In this section, we cover a few of them and aim to make the reader aware of the contradictory opinions in this regard, while reporting our performance score in both MRR and NDCG to avoid any kind of bias.

MRR: Although MRR is widely used in almost all IR benchmarks, it shows strange behavior in certain cases. For instance, assume that we have three queries, and system A returns the first relevant documents at ranks 1, 2, and 4. On the other hand, system B returns the relevant answers in each case at rank 2. For system A, we find the relevant item on average at rank $\frac{1+2+4}{3} = 2.33$, which is

worse than rank 2 for system B. Nonetheless MRR(A)=$\frac{1}{3}(1+\frac{1}{2}+\frac{1}{4}) = 0.58$ and MRR(B)=0.5. This is due to the property of the expected value, where $E(1/r) \neq 1/E(r)$. Fuhr (2017) points out this weakness of the MRR metric but also attributes the major problem of MRR to its ordinal scales and that the average of an ordinal scale is meaningless. However, there is an ongoing debate about whether RR is truly an ordinal scale or if it should be considered an interval scale (Sakai, 2020). One point worth mentioning though, is that although the behaviour of the MRR on systems A and B might seem strange at first, it aligns perfectly with the assumption of the metric. MRR relies heavily on the first relevant document and thereby rewards system A for the first query, hence the higher score. Optimizing for such measures is useful for instance in navigational search and it is complementary for other measures such as NDCG.

MAP: Fuhr (2017) mentions that MAP assumes that the user ends the search after the first relevant document is found and the probability of stopping is the same for all relevant ranks. While the first assumption might hold in some cases, the second assumption is unrealistic. Most users stop early on and do not continue down a long list of relevant results (Granka et al., 2004).
Zobel et al. (2009) also argue against both AP and NDCG, as these measures depend on the total number of known relevant documents, which is not always available.
Both Zobel et al. (2009) and Fuhr (2017) recommend *Ranked Biased Precision (RBP)* as an alternative to MAP (Moffat and Zobel, 2008). However, RBP is also not perfect and there are some discussions regarding its fundamental assumptions (Sakai, 2020). On the other hand, there exist major points that support the validity of AP. In particular, AP was identified as a reliable measure for the evaluation of systems in TREC (Buckley and Voorhees, 2005) and was useful in identifying effective term weighting schemes for models such as BM25 (Robertson and Zaragoza, 2009). Nonetheless, if there are graded relevance assessments it is best to report the result in a more stable metric such as NDCDG.

In summary, when choosing a metric for evaluation it is important to be aware of the implications and assumptions of each metric and also report multiple ones to not have a narrow evaluation based on a single criterion.

5.4.5 Ranking Performance

Table 5.4 shows the ranking performance of quantity-aware models, in terms of P@10, MRR@10, NDCG@10, R@100, and latency in milliseconds. The three models with a "Q" prefix indicate the disjoint quantity-aware rankers combined with BM25 as QBM25, SPLADE as QSPLADE, and with ColBERT as QColBERT. Neural models with a $_{ft}$ postfix are joint quantity-aware rankers fine-tuned on augmented data, namely, SPLADE$_{ft}$ and ColBERT$_{ft}$.

Permutation re-sampling is used to test for significant improvements over the baseline models (Riezler and III, 2005). Results denoted with † mark highly significant improvements over the baseline models, without quantity awareness with a p-value < 0.01. For instance, QSPLADE and SPLADE$_{ft}$ are compared with SPLADE for significance testing.

Table 5.4: P@10, MRR@10, NDCG@10 and R@100 on FinQuant and MedQuant. Top-2 values in each column are highlighted in bold and significant improvements over baselines are denoted by †.

	Model	latency (ms)	FinQuant				MedQuant			
			P@10	MRR@10	NDCG@10	R@100	P@10	MRR@10	NDCG@10	R@100
baselines	BM25	9	0.06	0.14	0.09	0.47	0.04	0.11	0.07	0.37
	BM25$_{filter}$	9	0.14	0.32	0.25	0.60	0.08	0.19	0.15	0.48
	Cohere$_{v3}$	-	0.14	0.22	0.19	0.27	0.10	0.17	0.15	0.25
	SPLADE	26	0.10	0.24	0.19	0.53	0.11	0.25	0.20	0.58
	ColBERT	36	0.15	0.35	0.27	0.70	0.12	0.31	0.24	0.63
disjoint	QBM25	311	0.21	0.53	0.41	0.55	**0.18**	0.47	**0.37**	0.51
	QSPLADE	319	**0.29**†	**0.67**†	**0.53**†	**0.83**†	**0.19**†	**0.52**†	**0.38**†	0.70†
	QColBERT	42	**0.30**†	**0.69**†	**0.56**†	**0.87**†	**0.18**†	**0.51**†	**0.37**†	**0.73**†
joint	SPLADE$_{ft}$	26	0.21†	0.51†	0.41†	0.74†	0.14†	0.37†	0.29†	0.63†
	ColBERT$_{ft}$	36	0.23†	0.55†	0.44†	0.77†	0.18†	0.44†	0.36†	**0.72**†

Both the joint and disjoint quantity ranking improves the quantity understanding of the IR models. However, contrary to our initial hypothesis, disjoint quantity rankers consistently outperform joint models across all metrics. The improvement is consistent across the precision-focused metrics of P@10 and MRR@10 as well as NDCG@10. The disjoint rankers show significant improvements, exceeding 10 points in P@10 and R@100, and over 30 points in MRR and NDCG compared to the base models (without the "Q" prefix). The joint models are slightly behind the disjoint ranker but also demonstrate enhancements, with around 10 points in P@10 and R@100 and over 15 points in MRR and NDCG compared to the base models.

Higher scores of the quantity-aware models in P@10 and MRR@10 indicate that the addition of quantity scores or task-specific fine-tuning pushes the relevant documents to higher ranks and to the top of the candidate list. The improvement in the NDCG score from the base models shows that the entire ordering of results (even in lower ranks) is better with the quantity-aware models. Moreover, the improved R@100 also indicates that more relevant items are included in the top 100. It is worth noting that the enhancement in the case of the disjoint quantity-aware rankers is achieved without requiring additional fine-tuning. The only drawback is a minimal increase in latency, especially for QBM25 and QSPLADE, where the quantity score is added to the entire ranking. This overhead diminishes for the top-performing disjoint ranker, QColBERT, where the quantity score serves as a re-ranker on the top-k candidates. ColBERT (base model without quantity scoring) shows a high recall on both datasets, suggesting that relevant results are within the top-k but not necessarily at the very top. Hence, the re-ranking with the quantity score proves beneficial.

Table 5.5: MRR@10, NDCG@10, and R@100 for the two evaluation datasets, where the fine-tuned models are trained using the other dataset as training set.

Model	FinQuant				MedQuant			
	P@10 ↓	MRR@10↓	NDCG@10↓	R@100↓	P@10↓	MRR@10↓	NDCG@10 ↓	R@100 ↓
SPLADE$_{out}$	-0.03	-0.06	-0.07	-0.04	-0.02	-0.01	-0.04	-0.05
ColBERT$_{out}$	-0.03	-0.07	-0.06	-0.03	-0.02	-0.01	-0.03	-0.02

The results for the joint quantity-aware rankers demonstrate that task-specific fine-tuning yields a performance boost. Although the metrics fall short of those achieved by the disjoint ranker, they still show significant improvement over the base models. This supports our hypothesis that the lack of task-specific data has exacerbated the challenge of quantity understanding for retrieval systems. In terms of latency, disjoint ranker perform as well as their baselines, since fine-tuning does not require any architectural change. Here, once again the ColBERT$_{ft}$ variant shows superior performance. ColBERT (base model without quantity understanding) also exhibits superior performance among the baselines. We attribute the better performance of the ColBERT-based model to the fine-grained token-level interactions that allow the model to learn better associations between tokens. In quantity ranking, token interactions play a more significant role compared to the query and document expansions conducted by SPLADE. This also showcases that the architecture and how the inter-token interactions are modeled matter for quantity understanding. Nonetheless, after manually looking at the cases where the neural models make mistakes, we noticed that even after fine-tuning, understanding numerical conditions remains a challenge. We investigate how much the fine-tuned models rely on quantities for ranking in Section 5.4.7.

5.4.5.1 Cross-dataset Generalization

The base checkpoint used in the construction of disjoint quantity-aware rankers is trained on general-purpose data without any additional fine-tuning specific to the financial or medical domains. Therefore, the results reported in Table 5.4 already show the performance on out-of-domain data for the disjoint rankers.

On the other hand, the joint quantity-aware models underwent task-specific fine-tuning on quantity-centric queries specific to either the financial or medical domains. It is worth investigating whether task-specific fine-tuning in one domain can be effectively transferred to another domain. Table 5.5 shows the performance drops of joint rankers on out-of-domain data, The negative values show the difference in performance compared to models fine-tuned on data generated from the same domain. Each model is fine-tuned on data from the other dataset and evaluated on a new domain. In general, both SPLADE and ColBERT show a minimal performance drop, with the data trained on the larger set, FinQuant, being more robust to domain change. This suggests that the models learn patterns for quantity-centric queries during the fine-tuning without memorizing queries in a certain domain.

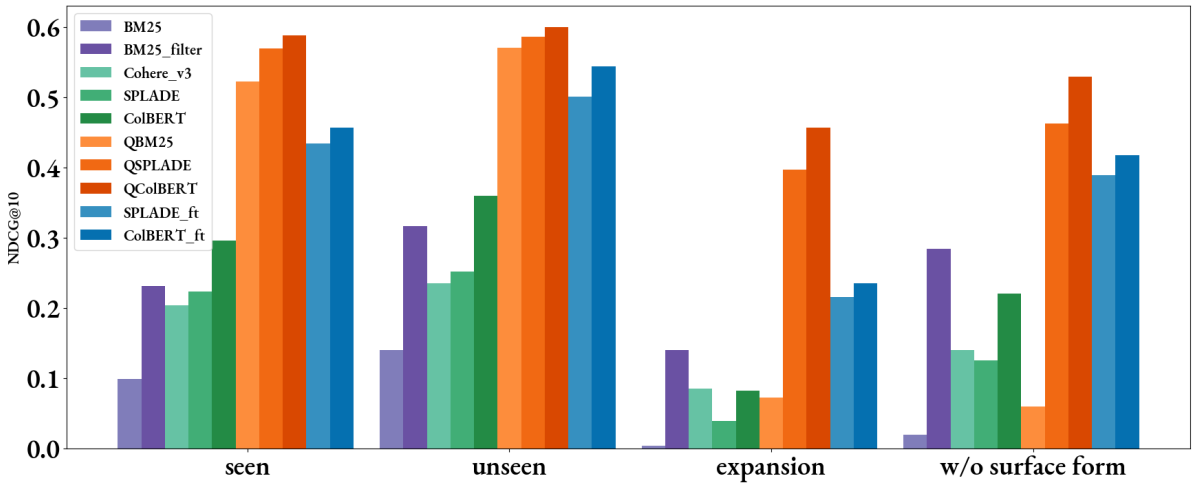


Figure 5.13: Performance in terms of NDCG@10 on different subsets of the test set, the *seen* and *unseen* category focus on lexical matching, whereas the *expansion* and *w/o surface form* are paraphrasings without exact match.

5.4.5.2 Performance on Lexical vs Semantic Queries

As outlined in Section 5.4.2, FinQuant and MedQuant contain a variety of lexical and semantic queries. *Seen* and *unseen* subset are keyword-based queries and *expansion* and *w/o surface form* contain instances of semantic queries. We evaluated the models on these different subsets in terms of NDCG@10. Other metrics show a similar pattern, however, the NDCG metric computes the performance on the entire ranking and is a better fit for the comparison of the systems.

The NDCG@10 scores of all models on the FinQuant dataset are illustrated in terms of a bar plot in Figure 5.13. The same pattern is observed for the MedQuant dataset, therefore, the plots are not presented here. The bars are color-coded to represent different model types: blue shades indicate the joint rankers, red and orange shades denote the disjoint models, green shade represents the neural baselines, and purple shade signifies the lexical baselines.

In the context of lexical models relying on BM25, the difference between semantic and lexical subsets is evident. These models show great performance on *seen* and *unseen* subsets but not on the semantic subsets. It is crucial to emphasize that the *expansion* subset involves queries from *seen* and *unseen*, paraphrased to include synsets and synonyms. Notably, the lexical models struggle to retrieve accurate results in this scenario.

Neural models also perform better on keyword-based queries. However, the decline in performance, when compared to lexical models, is less pronounced. A notable observation is that the combination of neural models with a disjoint quantity index does not impede their capacity for semantic retrieval.

5.4.5.3 Complexity of Numerical Conditions

The FinQuant and MedQuant datasets contain an equal number of queries for each quantity condition. In this section, we look at how performance changes for different numerical conditions.

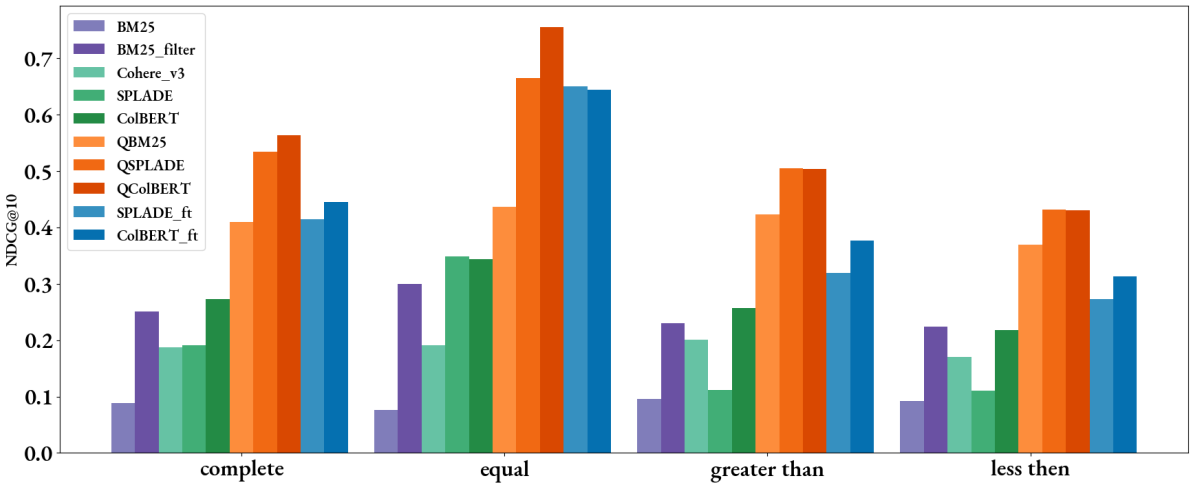
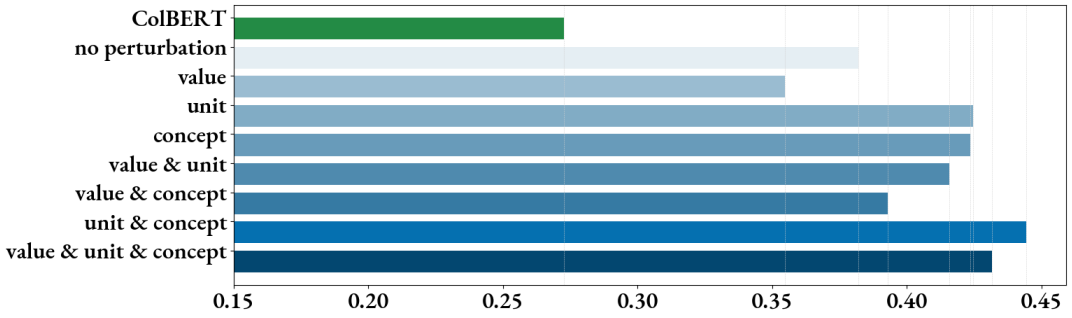


Figure 5.14: Performance in terms of NDCG@10 on different subsets of the test set, where each subset corresponds to queries containing a certain numerical condition.

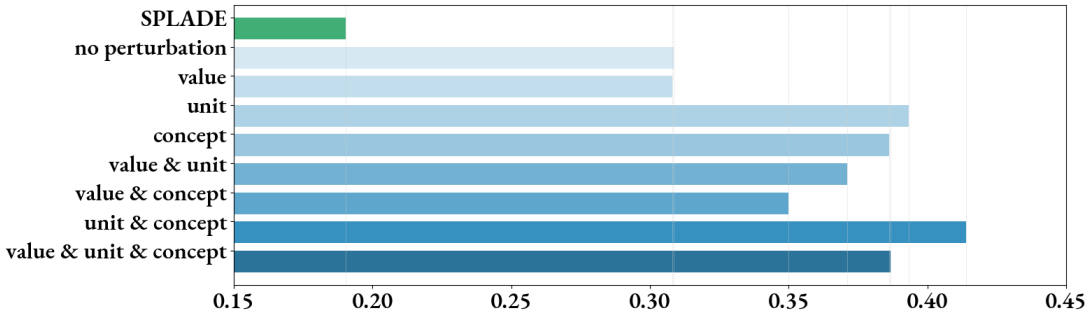
Figure 5.14 shows the NDCG@10 score of all models on subsets of various numerical conditions in comparison to the complete test set (all subsets combined) for the FinQuant dataset. Results on the MedQuant follow a similar pattern and are not presented here.
Equality queries are in general easier for the models as the notion of relevance in this case aligns with term-based ranking and finding exact matches for values is easier than interpreting bounds.
There is a substantial drop of almost 20 points in performance for the bound-based conditions in comparison to the equality queries. Interestingly, the subset containing less than condition shows a slightly greater drop in performance in comparison to the subset with greater than condition. The decline in performance for bound-based conditions is consistent across all models, implying that these conditions are harder for all models to rank. One hypothesis for this behavior is that the bound-based conditions require an understanding of scales, numerical comparison, and filtering based on hard numerical conditions, which are typically harder. Moreover, unlike the equality condition where exact token matching might help in retrieving the correct result, the relevant result contains ranges for values that do not match the query value, making the task relatively harder.

5.4.6 Ablation Study on Augmentation Methods

The performance of the joint rankers relies on the data generated for fine-tuning. While we provided justification for our generation mechanisms in Section 5.2.1, it is crucial to investigate the impact of various augmentation strategies and identify the most beneficial one. To this end, we perform an ablation study on generation techniques purposed.
We separate each strategy, and fine-tune the neural models (ColBERT and SPLADE) on data generated using a specific strategy or pair-wise combinations. The goal is to identify the optimal data generation strategy and evaluate whether incorporating synthetic contrastive examples enhances learn-



(a) ColBERT$_{ft}$



(b) SPLADE$_{ft}$

Figure 5.15: Ablation study on different augmentation methods, where *value* and *unit*, corresponds to value and unit permutation and *concept* corresponds to concept expansion.

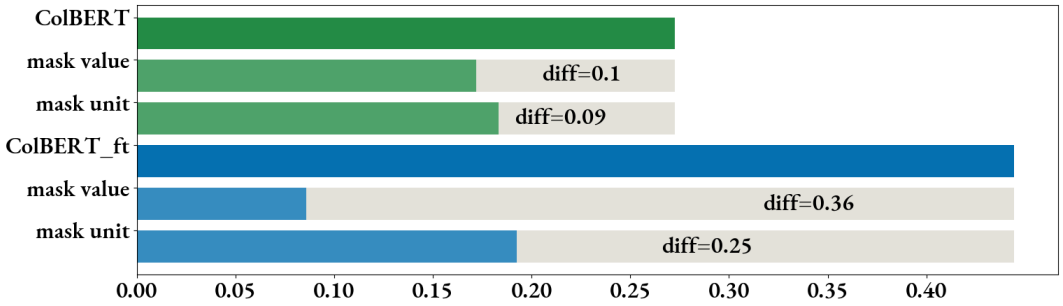
ing cues compared to using only positive and negative samples from the data.
There are three points of variability in data generation:

1. *Concept expansion* for semantic understanding in the query generation.
2. *Value permutation* for understanding of numerical conditions.
3. *Unit permutation* for understanding the semantics of units.

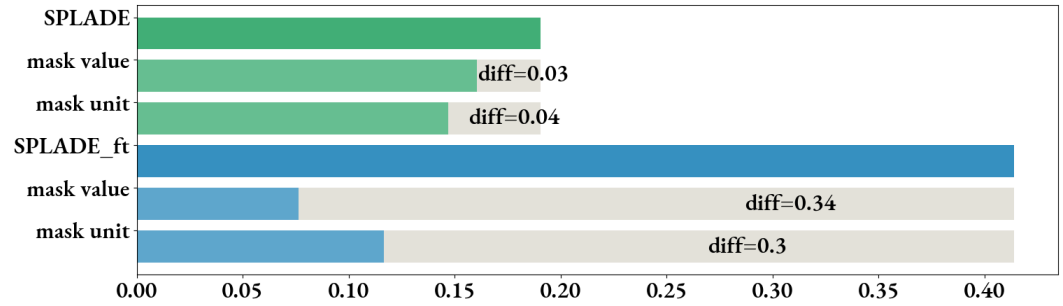
Once again, we present the result of the ablation study using the larger dataset of FinQuant, where the training data is generated from the financial domain and evaluated on FinQuant. The presented patterns are similar for MedQuant dataset.
The results for ColBERT$_{ft}$ and SPLADE$_{ft}$ are demonstrated in terms of NDCG@10 in Figures 5.15a and 5.15b, respectively. The NDCG@10 for the base checkpoint without any fine-tuning is added to the top of the figures in green for comparison. *no perturbation* corresponds to the case where no data augmentation was applied and only the positive and negative samples from the original sampling are used for training and the queries are not expanded using concept expansion.
In general, all the generation strategies enhance the performance of the base model, when used in isolation. However, contrary to our original hypothesis, combining all strategies does not yield the best-performing model.

An interesting observation is the detrimental effect of value permutation. The value permutation on its own enhances the performance of the base model. However, as soon as it is accompanied by other augmentation methods the performance degrades slightly. The best combination for both the SPLADE and ColBERT models is unit permutation and concept expansion. It is worth noting that both of these augmentation techniques on their own also provide a larger boost in comparison to value permutation. We find this behavior rather surprising and counter-intuitive. Usually, the performance of neural models increases with the amount of data presented for a given task, however, perturbing the values does not seem to enhance the performance as expected. This can be related to the internal representation of the neural models for numerical value, which is hindering their ability to correctly learn the semantics of them.
Due to the superior performance of unit permutation and concept expansion, the variant of the models presented for evaluation as ColBERT$_{ft}$ and SPLADE$_{ft}$ are trained on data generated with these strategies and value permutation is disregard.

It is crucial to note that we did not disregard the value permutation issue without thoroughly investigating its cause and attempting to alleviate the problem. Our initial assumption was that numerical conditions are more challenging to learn and therefore require more samples. However, increasing the sample size in the case of value permutation did not improve the results.
We also experimented with alternative ways to select the query value, other than the ones mentioned in Section 5.2.1. These methods included selecting the query value randomly or adding perturbations to the original values in the distribution. For instance, if the query value "100" is chosen for a condition, a random value is added or subtracted. Despite these efforts, the performance of the value permutation strategy did not improve, leading us to ultimately disregard them.
In an attempt to enhance the representation of numerical values, we experimented with the conversion of the values into scientific notation. Previous works (Thawani et al., 2021; Zhang et al., 2020) have shown promising results for quantity-aware embedding models, when the values inside the corpus are represented in scientific notation. Specifically Zhang et al. (2020) trained a quantity-aware BERT model by transforming values in the corpus into a combination of an exponent and mantissa. As an example, 314.1 is represented as 3141`[EXP]`2, where `[EXP]` is a new token added to the vocabulary. The authors claimed that the separation of exponent and mantissa allows the models to compare the scales of values more easily by simply attending to the exponent.
We applied the same procedure to all sentences in the corpus and the values in the queries during fine-tuning. However, contrary to previous findings, the conversion reduced the performance of the baseline models. It is essential to note that the base BERT model used in ColBERT and SPLADE was not pre-trained to interpret numbers in scientific notations, and these models had not encountered such notations during training in the general-purpose retrieval setting. Given the relatively small size of our dataset, introducing this new notation unexpectedly not only failed to enhance the model's performance but also introduced confusion.



(a) ColBERT based models



(b) SPLADE based models

Figure 5.16: The effect of task-specific fine tuning on models attention to quantity tokens. In the masked variants either the unit or the value of the sentences in the collection is masked.

5.4.7 Effect of Fine-tuning

In the disjoint quantity-aware ranking, separating quantity ranking and term ranking provides a clear understanding of how quantity scoring influences the ranking. On the contrary, in the joint quantity-aware models, it is not directly visible how the task-specific fine-tuning has affected the internal ranking strategy of the neural models. Since directly examining changes in dense query and sentence representations is not particularly insightful, we instead focus on the effect of tokens related to quantities in ranking. A quantity-aware model should learn to focus on tokens related to quantities while ranking a quantity-centric query, whereas a general-purpose model would focus on other tokens to find topical similarity between the query and a sentence.
To this end, we evaluate both the base models (ColBERT and SPLADE) and the fine-tuned models (ColBERT$_{ft}$ and SPLADE$_{ft}$) on two masked versions of the test data. We compare the base version of the neural models with their fine-tuned version on the different maskings of the FinQuant dataset. The results for the ColBERT models are shown in Figure 5.16a and for SPLADE models in Figure 5.16b. The masked versions include: masked value and masked unit.

Mask value: In this scenario, we mask all values in the collection with the `[MASK]` token before running the evaluation. As a result, the model can no longer use value comparison to retrieve the correct

result. This task aims to determine the extent to which the model depends on the value token.

Mask unit: Here, we mask unit tokens in the collection before running the evaluation with `[MASK]` token. This task is intended to observe the impact of unit comparison on the final ranking.

The fine-tuned version exhibits a more significant drop in performance compared to the base models when quantity tokens are masked. This indicates that after fine-tuning, the model becomes more dependent on tokens related to quantities (values and units), to identify the relevant sentence.

5.4.8 RAG-based Evaluation

Despite the history of the machine learning model being heavily driven by manually labeled data, as large language models became more effective the techniques have shifted towards automatic evaluation with zero-shot prompting. In such cases, a language model is prompted with a template like

```
Provide a rating on a scale of 1 to 10 of whether these search results are relevant
to the query. The query is {query} , and the search results are {search_results}
```

A language model would output a score that is used as a relevance score during the evaluation of the system. The most notable framework is *Retrieval Augmentented Generation Assessment (RAGAS)* (ES et al., 2024). The RAGAS score is made of a variety of prompts for evaluation of the generation and retrieval. There are other extensions to the RAGAS method, e.g., Saad-Falcon et al. (2023) propose that training a language model as an evaluator has a better performance than zero-shot prompting for a score.
To evaluate our quantity-aware RAG system we utilize the RAGAS framework, to check if the generated answers for quantity-centric questions are correct and align with the user's information needs. Other related metrics that can be used in this setting are those developed for factual consistency of summaries [22] for summarization models (Fan et al., 2023), but we do not cover those methods in detail and focus on the RAGAS framework only.
Since the retrieval component has already been evaluated in the preceding section using a more dependable method involving human annotations, we concentrate solely on metrics related to the answer generation. In the following, we outline selected metrics from RAGAS and highlight the ones we use in this study.

5.4.8.1 Metrics

The RAGAS metrics cover both retrieval and generation. Here, we only discuss *faithfulness* and *answer relevance*. Metrics such as *context relevance*, *context precision*, and *context recall* focus on the

[22] Generating an answer from multiple context snippets is similar to multi-document summarization. Given that we are handling quantities and their values to meet specific numerical conditions, metrics for factual consistency are particularly relevant.

quality of retrieval and are not relevant to the final generated output. Additionally, we exclude *answer relevancy*, since it primarily evaluates whether answers are generated appropriately, regardless of factual accuracy. This metric penalizes incomplete or redundant information, which does not align with the focus of our evaluation. We are not interested in the quality and fluency of generation (which is more related to the language model) but are more interested in how the retrieval model impacts the generation output.

Faithfulness: To avoid hallucinations, and to ensure that the retrieved context can act as a justification for the generated answer, faithfulness is introduced. An answer is faithful to the context if the claims that are made in the answer can be inferred from the context. To estimate faithfulness, a language model is used to extract a set of statements or claims from the generated answer. Then each one of these claims is checked with a given context to determine if it can be inferred from the context or not. The faithfulness score F is computed as shown in Equation 5.24, where $|V|$ is the number of statements that were supported by the context, and $|S|$ is the total number of statements.

$$F := \frac{|V|}{|S|} \tag{5.24}$$

It is worth noting that the claims extracted from the answer and the context are usually general facts and this score is not tailored to extract claims that are quantity-centric. To be more precise, consider the prompt used in the RAGAS framework. [23]

```
Given a question, an answer, and sentences from the answer analyze the complexity of each
sentence given under 'sentences' and break down each sentence into one or more fully
understandable statements while also ensuring no pronouns are used in each statement.
Format the outputs in JSON.
```

This prompt does not encourage the model to focus on quantities. However, we did not change the implementation of the RAGAS framework.

Answer correctness: This metric compares the generated answer and the ground truth (an answer written by human annotators). The score ranges from 0 to 1. A higher score indicates a closer alignment between the generated answer and the ground truth, signifying better correctness. The focus is on two aspects of *semantic similarity* (Risch et al., 2021) between the generated answer and the ground truth, as well as *factual correctness*. The final metric is computed as the average of factual correctness and the semantic similarity between the given answer and the ground truth. To compute the semantic similarity both the answer and the ground truth are vectorized and their cosine similarity is computed as shown in Equation 5.25, where τ is the vectorization function.

$$\text{semantic} := cosine(\tau(\text{answer}) \cdot \tau(\text{ground truth})) \tag{5.25}$$

[23] Taken from the repository for RAGAS: https://github.com/explodinggradients/ragas/blob/main/src/ragas/metrics/_faithfulness.py (last accessed: 02.05.2024).

Factual correctness quantifies correctness based on the number of facts in the ground truth and the generated answer and it is shown in Equations 5.26. The facts are once again extracted using a language model. TP (True Positives) counts the number of facts present in both the ground truth and the generated answer. [24] FP (False Positives) counts the facts present in the generated answer but absent in the ground truth. FN (False Negatives) counts facts present in the ground truth but missing in the generated answer.

$$\text{factual} := \frac{TP}{TP + 0.5 \times (FN + FP)} \tag{5.26}$$

Similar to faithfulness this metric is general purpose and does not focus on quantity facts.

Factual consistency score: The metrics from the RAGAS framework are not trailered towards quantities and we are already aware of the limitations of the language models in this regard. Therefore, we introduce an additional metric based on human annotation to account for a more objective view of the generated output. In this case, the annotator is presented with the generated answer and ground truth and is asked to rate the answer between 0 and 1, based on the following guidelines:

- A score of 1 is assigned to answers that cover all the facts in the ground truth correctly.
- A score of 0 is assigned to answers that contain contradictory facts or none of the facts mentioned in the ground truth.
- A score of 0.5 is assigned to incomplete information, where the generated answer is partially true but does not cover all the facts.

The average of the scores across all test instances shows the factual consistency score. This score is similar to the factual correctness score from RAGAS but relies on human judgment instead of a language model inference. As a result, we can make sure that the focus is on quantity facts. [25]

5.4.8.2 Dataset and Evaluation Setup

To assess the impact of quantity-aware retrieval on subsequent generation within a RAG pipeline, we manually create a set of 42 question and answer pairs. The questions are crafted from the same news articles that were utilized to create the FinQuant dataset for evaluation of the rankers. The questions are created in such a way that they contain a quantity and a numerical condition, e.g., "What are two car models from Tesla that cost more than 71k US dollars?" or "Which Apple watches are less than 42 millimeters, what are the prices?". The dataset is evenly divided across three numerical conditions (equal, greater than, less than), with 14 questions allocated to each condition.

[24] Not to be confused with tp, fp, and fn in retrieval, although the semantics are similar.

[25] Such annotation is cumbersome in the case of a large test set, however, we have a small set of 42 questions and manually comparing the results is feasible in this case. For a large set of examples, one can rely on automatic quantity fact extraction by replicating methods similar to Ho et al. (2019).

The ground truth answers were crafted by the author of this thesis after carefully reading the top-10 retrieved contexts by three retrieval models used in this evaluation (ColBERT, QColBERT, and ColBERT$_{ft}$). Therefore, the ground truth answer includes the knowledge of all three rankers. If a correct context is not in the top-10 result of any of the rankers, it will not contribute to the final answer, which should not matter for this specific evaluation setup as we are only considering the named rankers. The factual consistency score is manually added after examining the generated answers based on the guidelines described in the previous section. Once again the annotations are performed by the author of the thesis.

In certain instances, there exist questions with multiple valid answers. For example, "Name at least three smartphones in the context provided with 48MP cameras?". Depending on the provided context a variety of smartphones can satisfy this constraint. In such cases, the ground truth answer encompasses the names of all smartphones with 48MP cameras mentioned in the top-10 contexts of ColBERT, QColBERT, and ColBERT$_{ft}$. Then, for the factual consistency score, if the generated answer contains three of *any* of the smartphones in the list, the answer is marked as correct. However, if only two of them have 48MP cameras and the last one does not satisfy the constraint, the factual consistency score drops to 0.5, indicating a partial match.

Since our focus is on frozen RAG, no further training for aligning the generator and the retriever is conducted. Sentences within the corpus are ranked utilizing one of the evaluated rankers (ColBERT, QColBERT, and ColBERT$_{ft}$), and the surrounding context along with the question is supplied to a language model for answer generation. We used the `gpt-3.5-turbo` model from Open AI for generation and set the temperature to 0 for a more consistent output (other parameters we left to the default setting). Nonetheless, we still observed a lot of variability in the generation provided the same context. We also experimented with multiple prompts inspired from LangChain [26] and LlamaIndex [27] frameworks for RAG and settled on the one that worked best for `gpt-3.5-turbo`, since the variability in prompt has an impact on the output (Mizrahi et al., 2024). The generation prompt is as follows:

```
System: Use the following pieces of contexts and their associated dates to answer the
user's question. If you don't know the answer, just say that you don't know, don't
try to make up an answer. Be short but comprehensive, and use numbered bullet
points when necessary.
----------------
Context: {context}
Date of the context:{context_date}

Given this information, please answer the question:
{question}
```

The placeholders {context} and {context_date} are filled using the information from the ranker. Here, {context_date} represents the publication date of the article from which the corresponding

[26] https://www.langchain.com/ (last accessed: 02.05.2024).
[27] https://www.llamaindex.ai/ (last accessed: 02.05.2024).

context is selected. The date becomes important for answering questions such as "When was the first instance of Germany's yield dropping less than 3.5 bip in the year 2019?". The date information is part of the metadata of the articles. The {context} placeholder includes the paragraphs of the highest-ranking sentences and the paragraphs before and after, retrieved from the hierarchical index. To compute the evaluation metric we rely on the `ragas` package. [28] The package uses `open-ai-embeddings` [29] for the computation of semantic similarity and and `gpt-3.5-turbo` for factual correctness and faithfulness.

5.4.8.3 Evaluation Results of RAG

Table 5.6: Faithfulness, answer correctness, and factual consistency on the quantity-centric question answering dataset for ColBERT models with and without quantity-awareness. The best-performing values are marked in bold.

	faithfulness	answer correctness	factual consistency
ColBERT	0.63	0.57	0.67
QColBERT	**0.78**	**0.59**	**0.71**
ColBERT$_{ft}$	0.73	0.48	0.60

The evaluation results based on faithfulness, answer correctness, and factual consistency are shown in Table 5.6. Since the generation is non-deterministic and we observed great variability in the outputs, the results are averaged across three distinct runs. For each question, the language model is provided with the top-10 context paragraphs to ensure sufficient context for generated answers. As demonstrated by the result, the best performing quantity-aware model (QColBERT) also facilitates more accurate and factual consistent generated answers. Interestingly, the answer correctness and factual consistency have lower values for ColBERT$_{ft}$ when compared to ColBERT, despite the retrieval outcomes suggesting the contrary. Nonetheless, providing the correct context is not sufficient for the language model to generate the best answer. One notable pattern observed is that despite the retrieved context meeting a numerical condition, the language model's reasoning process may still yield inaccurate output or if the retrieved context does not meet the numerical condition, the language model ignores the condition altogether. For example, for the question "Give me information for leases in the context provided for more than $550 per m2.", the language model generates the following response given the context from ColBERT:

```
The following leases have rental rates of more than $550 per sq m:

- In the context 2019-05-21, a private accounting firm leased the ground floor at 302-320
Burwood Road for $523 per sq m.

- In the context 2020-04-07, NPCo leased level 2 at 103 Lonsdale Street for $650
gross per sq m.
```

[28] `ragas`: https://docs.ragas.io/ (last accessed: 02.05.2024)
[29] https://platform.openai.com/docs/guides/embeddings (last accessed: 02.05.2024)

```
- In the context 2019-11-26, Device Technologies Australia leased a 1581 sq m office
at 6-8 Compark Circuit for $280 per sq m.

- In the context 2019-05-21, a vacant two-level commercial building at 177 Burnley
Street was leased to accounting firm BR Wellington Pty Ltd for a commencing annual rental
of $320,000 net plus.

These are the leases with rental rates exceeding $550 per sq m in the provided contexts.
```

In this scenario, 3 out of 4 leases presented in the answer fail to meet the constraints of the numerical condition, despite the model ultimately reasoning that they do (last generated line). In the same manner, even when the correct context is present, at times, the language model fails to distinguish the correct information to incorporate in the final answer.
Implementing techniques like chain-of-thought prompting (He et al., 2023) might enhance the model's capability in such instances. However, we did not experiment with alternative methods.

5.5 Community Support

To showcase the performance and easy integration of disjoint quantity-aware retrieval, we developed and open-sourced an Elasticsearch [30]/OpenSearch [31] plug-in for the lexical model of QBM25. [32] To demonstrate the plug-in, we developed an online interface called *QFinder*, which is accessible at https://qfinder.ifi.uni-heidelberg.de/. Since Elasticsearch and Opensearch are used frequently in the industry, this plug-in allows for easy integration into popular open-sourced search engines.
The only difference to QBM25 used in quantitative analysis of the rankers is that quantulum3 library [33] was used for quantity extraction instead of CQE. Quantulum3 has limited functionality in comparison to CQE (refer to Section 4.4 for more details on this topic) and cannot detect numerical conditions. Therefore, the QFinder interface requires a strict query language, in which the user has to specify search terms, numerical conditions, values, and units separately. Another notable difference is that queries with ranges are also supported.
The underlying data for the interface is a subset of the FinQuant dataset. The subset contains 186,614 English news articles from January to December 2021 on the following subjects: economics, sports, technology, cars, and companies. In the upcoming sections, we delve into the implementation details of both the backend and the frontend. Then, the interface functionalities are demonstrated by walking through the components and describing three query scenarios.

[30] https://www.elastic.co/ (last accessed: 02.05.2024)
[31] https://opensearch.org/ (last accessed: 02.05.2024)
[32] https://github.com/milenabruseva/qfinder-elasticsearch-plugin (last accessed: 02.05.2024)
[33] https://github.com/nielstron/quantulum3(last accessed: 02.05.2024)

5.5.1 Backend Implementation

Since the retrieval units are sentences, we split all articles into sentences using SpaCy 3.9 [34], resulting in a total of 6,599,355 sentences, where 466,632 sentences (around 14%) of the corpus contain quantities. As mentioned, to detect and normalize quantities and their units, we used the quantulum3 library. Each sentence, along with its list of extracted units and values, is added to an Elasticsearch index as an individual document. The reference to the original news article is kept for the retrieval of context information. The core of our ranking is performed by QBM25 implemented as an Elasticsearch plug-in. By developing our plug-in, we take advantage of this scalable solution for full-text search and extend it further for quantities in unstructured texts. The backend is based on a single-node instance of Elasticsearch and on it the QBM25 plug-in is installed. For a multi-node cluster, the plug-in has to be installed on every node. The plug-in itself extends the *ScriptPlugin* class and implements its own *ScriptEngine*. The ranking is done in two stages. First, the maximum BM25 score of query terms is computed to compute the normalized BM25 score for term ranking. Second, the normalized BM25 score is combined with the quantity score for the final ranking. The plug-in is developed in Java and is realized on Elasticsearch version 7.16.3.

5.5.2 Frontend Implementation

The web server is realized on top of the Python Flask [35] framework, and the interface is implemented using HTML/CSS and JQuery [36]. Communication between the user interface and server is built on AJAX and passes information in both directions as JSON objects. The Bootstrap [37] library is used for a responsive layout.

5.5.3 Interface Functionality

Query Formulation: The user input for quantity queries consists of *search terms*, a *quantity condition*, a *value* and a *unit*. All the mentioned values need to be provided in the designated input boxes as the system at this point cannot parse queries written solely in natural language. In the following, all the search components on the front page as shown in Figure 5.17 are described.

- The reset button next to “SEARCH” clears all input boxes.
- “Search terms” are the keywords of the query and consist of any term related to the query quantity. This can be a single term or a sequence of tokens. This box should only contain the terms associated with a quantity, to define values and units, the respective fields should be used.

[34] https://spacy.io/ (last accessed: 02.05.2024)
[35] https://flask.palletsprojects.com/en/2.0.x/ (last accessed: 02.05.2024)
[36] https://jquery.com/ (last accessed: 02.05.2024)
[37] http://getbootstrap.com (last accessed: 02.05.2024)

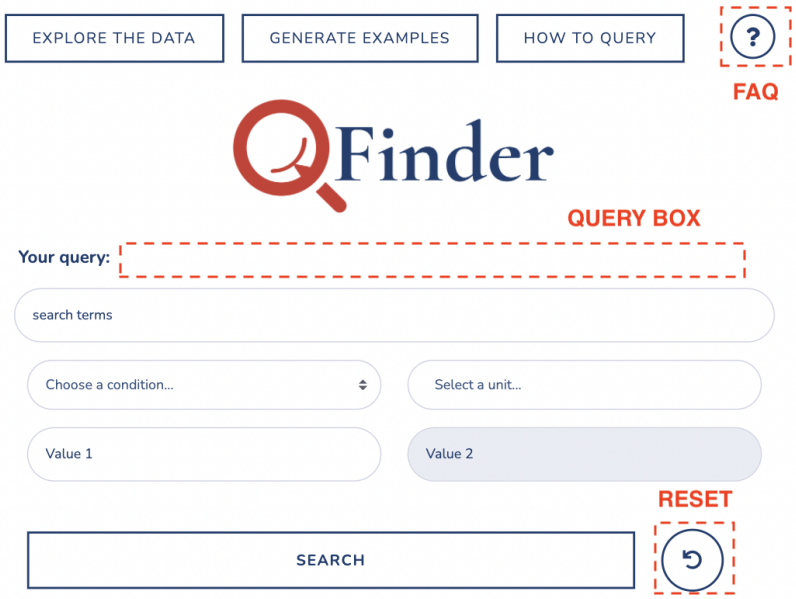


Figure 5.17: Overview of the QFinder user interface. A quantity query consists of *search terms*, a *quantity condition*, a *unit* and a *Value1* (or *Value2* as well for range conditions).

- Values (textboxes "Value1" and "Value2") and a unit define the quantity of the query with a numerical value and a specific unit. The numerical value should be provided in a standardized format, e.g., "1000" instead of "1K".
- The unit ("Select a unit..." dropdown) is chosen from a pre-defined set of units that were detected in the collection. The select box supports inline search and contains a list of all available units. Although the majority of well-known units used in the finance domain are covered here, this list is not comprehensive.
- From the "Choose a condition.." dropdown, users can choose between the four quantity conditions: equal, greater than, less than, and ranges. Single-value conditions are equal, greater than, and less than, meaning that they require only a single value. For ranges of values, upper and lower bound values should be specified in "Value1" and "Value2" text boxes.
- Upon filling in the input boxes, the user's query is formulated in natural language in the "Your query" box. This formulation is rule-based by concatenating the search terms with numerical conditions and the provided quantity.

Other Functionalities: QFinder's interface contains additional components, to help the user in query formulation and to answer frequently asked questions:

- The question mark on the top right corner guides the user to the FAQ page, with more information about the interface, corpus, and Elasticsearch plug-in.
- The "HOW TO QUERY" button triggers a step-by-step walk-through on how to use the system. It guides a new user through a sample of query formulation.

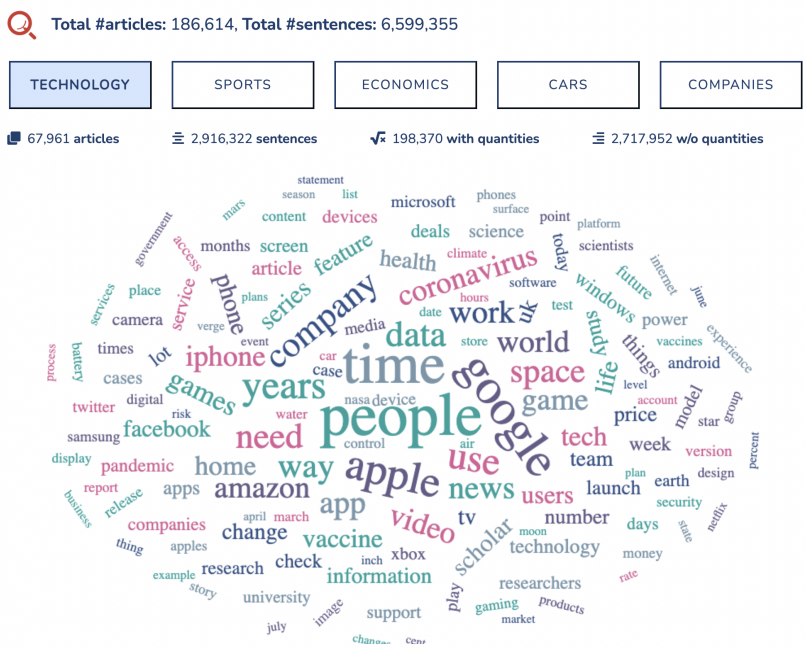


Figure 5.18: Exploring the corpus, where the word cloud shows the prominent topic in each category and the statistics of sentences and quantities are shown above it.

- As an example of how to formulate queries, we provide a set of pre-defined quantity queries. The "GENERATE EXAMPLES" button fills the input boxes with a random query from this pre-defined set, and the user can directly click on the "SEARCH" button to see the results.

- Despite a large number of articles in our collection, the topics are rather limited and if the user searches for a topic outside of the scope of the current collection, the results are not satisfying. To help users in query formulation, we provide a word cloud representation of each topic with statistics of the number of sentences and articles. The user can explore the underlying data upon clicking the "EXPLORE THE DATA" button. An example based on the topic "technology" is shown in Figure 5.18. Users can choose search terms that are related to the topics from the word cloud.

5.5.4 Query Scenarios

To better demonstrate QFinder's functionality, here we go through three typical search scenarios for (1) single value numerical conditions, (2) range, and (3) pure term-based search.

Search with Single Value Conditions: Equal, less than and greater than are all single value conditions, since they require a single numerical value for comparison against the query value. For such queries, "Value2" is greyed out and not used during ranking.
A sample query "mustang greater than 380 horsepower" for the quantity condition greater than is

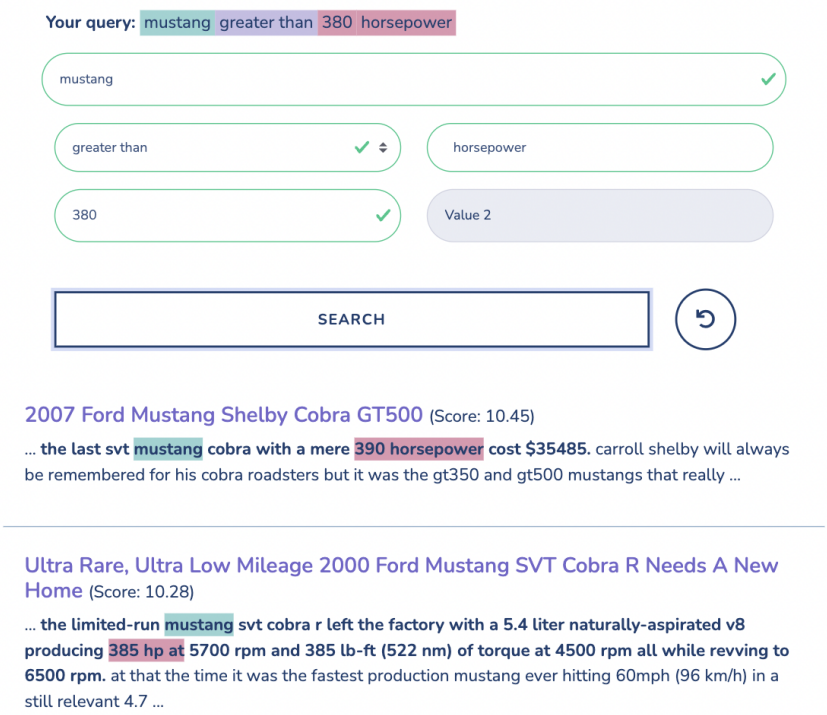


Figure 5.19: Single value condition on sample query "mustang greater than 380 horsepower".

shown in Figure 5.19. For brevity, we only show the top-2 results in the screenshot, but the interface contains the first 100 results and supports pagination. In each result set, the relevant sentence is highlighted in bold and the sentence after it is shown for more context information. The QBM25 score (combination of term-ranking and quantity-ranking) is presented in parentheses alongside the article title, and users can access the relevant article by clicking on the title. Additionally, if the search term or value and unit are present in the text, they are also highlighted and color-coded for visibility.

In Section 5.1.2, we discussed various quantity ranking functions for each numerical condition. The Elasticsearch plug-in published alongside the interface offers the option to choose between different sorting methods and heuristic functions. However, for presentation purposes in QFinder, we utilized the most intuitive sorting. For the equal condition, we used the descending exponential decay variant, where values close to an exact match are ranked higher. For the less than and greater than conditions, we applied ratio weighting with descending sorting, where values are sorted based on the proximity to the query value.

Quantity Search for Ranges: In the case of ranges, two values are required in the query, one for the lower bound and one for the upper bound. Upon selecting the range condition from the set of quantity conditions, the "Value2" input is enabled, allowing the user to specify an upper bound. In this setup, "Value1" serves as the lower limit for the range. Following the QBM25 ranking function, the sentences ranked at the top contain the query unit, meet the range condition, and encapsulate

Figure 5.20: Between (range) condition for sample query "unemployment rate between 3 and 5 percent".

values strictly falling within the specified range. An example of a range query as "unemployment rate between 3 and 5 percent", is shown in Figure 5.20.

As for the choice of the heuristic function, the ranges in QFinder are ranked based on the average of the bounds in descending order, where the values near the average of the lower and upper bounds are ranked higher.

Term-based Search: For a term-based search, the user is only required to provide "search terms". All the remaining inputs can stay empty. In this case, the ranking method falls back to the Elasticsearch default BM25 and quantity score do not have an impact on the final result.

5.6 Summary and Discussion

In this chapter, we (1) proposed two methods, joint and disjoint quantity-aware models, to integrate quantity understanding into both classical IR models as well as recent neural architectures, (2) introduced two novel benchmark datasets containing quantity-centric queries in domains of finance and medicine, (3) demonstrated significant improvements in quantity understanding over the baselines for both proposed techniques, and (4) open-sourced our code and data and provided an Elasticsearch/Opensearch plug-in for quantity-aware retrieval.
Our joint and disjoint approaches enable the integration of quantity understanding without altering existing architectures and with minimal overhead in query latency. We further highlighted the

strengths and weaknesses of both approaches, arguing that the choice of method should depend on the specific use case scenario. For the joint quantity-aware rankers, we summarize the strengths and weaknesses as follows:

1. The disjoint models consistently outperform the joint models across various domains.
2. The unsupervised and heuristic nature of the disjoint rankers makes them more flexible than the joint rankers. The notation of quantity proximity is easily altered by changing the quantity ranking function, leading to great flexibility in terms of different sorting of results.
3. The disjoint ranking can be combined with any lexical or semantic IR models without the change in their architecture or need for additional fine-tuning.
4. The quantity index and computation of quantity score introduce overhead in terms of query latency and make these models susceptible to errors from quantity extraction.
5. The independence assumption between the quantity and term ranking leads to a possible concept-quantity mismatch in the results.

For the disjoint quantity-aware rankers, we summarize the strengths and weaknesses as follows:

1. The joint models are better at finding concept and quantity associations as the quantity and term ranking are not separated by design (validated by error analysis and comparing the top results of joint rankers side by side with the disjoint rankers).
2. The joint approach eliminates the need for an external index and the associated efficiency overhead and errors in quantity extraction.
3. The overall performance of the joint models is lower than the disjoint variants.
4. There is a need for an additional fine-tuning step on synthetic data.

Depending on the user's needs, both approaches are viable options to enhance the quantity understanding of the current IR systems.
In addition to evaluating our models on two novel benchmark datasets in terms of retrieval performance, we highlight the importance and impact of quantity-centric ranking within recent question-answering architectures using an RAG pipeline, where the retriever is replaced by a quantity-aware model. Our findings show that using a quantity-aware model for retrieval in quantity-centric questions enhances the factuality and correctness of the generated answers, demonstrating the effectiveness of our approach for downstream applications.

To conclude, we also highlight the limitations of the proposed evaluation resources and the joint and disjoint quantity rankers.

Data: One immediate consideration about the datasets is the relatively limited number of test queries compared to larger-scale datasets such as MSMARCO (Nguyen et al., 2016). This is mainly due to limited human resources and budget in an academic setting. Nonetheless, we argue that this number of queries is already enough to showcase certain quantity-centric capabilities. Another shortcoming of the data is the absence of queries for (1) ranges, e.g., "iPhone with price between 500 and 800 dollars", and (2) negations, "iPhones not equal to 500 dollars".

Models: When considering neural models, one limitation is their reliance on hardware capabilities, particularly the need for GPUs, to ensure efficient training, indexing, and inference. The query latency values reported would suffer greatly if the computations were done on the CPU.
Another limitation is regarding the quantity extractor. Both the synthetic data generation paradigm and the disjoint quantity-aware models rely on a quantity extractor. In the case of the disjoint model, the quality of the quantity index directly relies on the quality of value and unit extraction. If a value and unit is not detected by the extractor it will not be considered by the ranking function. In the joint model, for data generation, the quantity extractor should also possess the ability to detect concepts in text, introducing the potential for additional error propagation through the system.
One modeling limitation is that we do not discuss models that deal with negations. Adding this variation to the disjoint models requires only a change in the numerical ranking function but it is more difficult for the joint setting where proper training data is required.
Finally, for the bound-based conditions of less than and greater than, we considered open bounds. Depending on the user intent closed bounds might be more appropriate, however, similar to the optimal sorting of results, this issue does not have a single solution.

6 Conclusions and Outlook

> "The end of a melody is not its goal: but nonetheless, had the melody not reached its end it would not have reached its goal either. A parable."
>
> *Friedrich Nietzsche*

Quantities are pivotal elements for communicating precise and clear information across various domains, enabling comparisons, analysis, and insights into trends and patterns. Quantity-centric retrieval, in which the system learns to prioritize quantities and their relations to textual elements for an effective ranking, becomes essential when the information needs of users involve constraints on quantitative attributes. Such systems should not only demonstrate an understanding of values and scales but also be able to capture relationships between quantitative data and other textual elements. In this final chapter, we summarize our contributions towards creating such a system, and discuss the advancements that quantity-aware retrieval and its extensions provide.

6.1 Summary and Contributions

In Chapter 1, we motivated the necessity for dedicated systems for quantity-centric retrieval and pointed out the shortcomings of current search engines concerning quantitative information in unstructured text. We highlighted the importance of dedicated models that are adept at handling numbers and scales and discussed the challenges encountered by systems when quantities are treated in the same way as other textual tokens.

In Chapter 2, we gave an overview of established methods in information retrieval, which creates the basis for our proposed methods in later chapters. This chapter follows the evolution of IR methods from the early probabilistic models to the recent neural architectures. We covered the traditional models, based on inverted indices and statistics of the corpus as well as modern language models and their impact on neural retrieval systems. For neural retrieval, we covered a variety of methods based on dense and sparse embeddings and also discussed prominent training strategies and efficient storage and retrieval based on approximate nearest neighbor search. By discussing the state-of-the-art architectures in retrieval and how relevance between queries and documents is defined and learned,

we identify the shortcomings of current systems in relation to quantity-centric queries.

Since none of the current retrieval systems focuses on quantities in text, in Chapter 3 we provided an overview of related work that directly or indirectly influences quantity representation in text. This includes methods for extraction of standardized quantities from unstructured text, a limited number of works that look at use-case-specific quantity-centric retrieval as well as methods to enhance quantity representations in language models and word embeddings. Since question answering is tightly related to retrieval, we also covered the cases where dedicated modules were implemented for quantity understanding and reasoning in reading comprehension tasks. Although many of these works do not focus directly on quantity-centric queries, they provide an overview of the effort made in the community to enhance quantity understanding in text.

Based on these considerations, we then explored various frameworks for quantity-centric information retrieval in the following chapters.

Extraction of quantities: In order to build a quantity-aware retrieval system, we first need to correctly identify all quantities in text. Therefore, as our first contribution, we introduce a comprehensive framework for the extraction and normalization of quantities in unstructured text. Upon reviewing the literature discussed in Chapter 3, we identified two major shortcomings in prior quantity extractors preceding our work. The first limitation involves restricting extraction solely to numerical values and a selected few scientific units. The second limitation lies in isolating quantities during extraction and neglecting crucial contextual information. Our proposed extractor alleviates some of these problems and allows for a more comprehensive quantity model.
We begin Chapter 4 by looking at various quantity types and their appearance in the text. We then define contextualized quantity as a unit and value pair, which is enriched with contextualized information. This includes concepts that are related to a quantity and how the value is changing. Since we do not want to study quantities in isolation, our focus is to go beyond the conventional representation of units and values for quantities by contextualizing them with other tokens within a given text. Our efforts in this direction led to:

I the creation of the first-ever quantity extraction benchmark and,

II the CQE extractor as a hybrid system of rules and a BERT-based classifier for unit disambiguation, which is able to handle a variety of quantity types, is easily extendable by adding new rules or new units to the dictionary and is not limited to a specific domain.

More importantly, CQE extractions enrich the corpus with quantitative information that can be used by retrieval models proposed in Chapter 5 for enhancing quantity-centric retrieval.

Quantity-aware retrieval systems: Building on the construction of CQE, in Chapter 5 we explored ways to enhance the quantity understanding of current retrieval systems. From the lessons learned in the efforts of others in developing quantity-aware language models and embeddings (as discussed in Chapter 3), our objective diverges from merely creating a single, specialized model for quantity-centric retrieval, which may lack applicability across application domains. The aim is to enhance current models such that they can handle quantity-centric queries as well as keyword queries. To this end, we first define quantity-centric queries as queries that contain numerical conditions. We propose two families of quantity-centric models. The models are distinguished based on whether the quantitative information is ranked jointly or independently from the textual content.

The disjoint model operates on the assumption that quantities have different semantics from textual tokens, and by combining their ranking objective into one, we are disregarding the correct notation of similarity based on scale and magnitude and replacing it with topical similarity. To this end, the disjoint ranker completely separates the ranking of the quantitative part of a query (value and unit pair with a numerical condition) from the textual part (search terms). Inspired by traditional inverted index structures, described in Chapter 2, the quantitative dimension of the corpus is encapsulated in a quantity index, where distinct unit and value pairs point to sentences containing them. A set of heuristic functions designed to capture numerical proximity under various numerical conditions operates on top of this index structure to handle quantity ranking. Due to this separation, the notation of similarity for the quantity part of the query can now deviate from the textual part. The term-based ranking can be performed by any neural or lexical ranker as long as the score is normalizable. Quantity-centric scores are then the combination of the term and quantity scores.

The main advantages of the disjoint approach are its flexibility, simplicity, and dispensing with training data. Nonetheless, the assumption of independence between quantity and term-based ranking is limiting and ignores the interdependency of the two. It would be more beneficial if one could learn both quantity proximity and textual similarity with a single objective and in a joint setting, conserving the inter-dependency between all token types.

Quantity-centric queries are not part of the training dataset for a general domain neural retrieval dataset. Therefore, we set out to experiment if the proper data is presented to these models, to what extent neural retrieval systems can capture the quantity and textual relations in their internal representations. It is important to note that we did not aim to create a dedicated model architecture, similar to the ones in Chapter 3, but rather a training paradigm that can enhance the quantity understanding of current prominent neural models. A dedicated quantity-centric model limits the application focus and is not useful in real-world scenarios, where users want to perform different types of queries.

Our training paradigm utilizes CQE to generate quantity-centric queries and positive and negative examples for contrastive learning. The examples include evidence for the model to learn relevance based on value comparison and unit understanding, which are the two main sources of errors in neural rankers. We observed that by adding an additional fine-tuning step on this synthetic data,

the quantity understanding of the neural models improved significantly. Yet, in our experiments the disjoint quantity-centric models still exhibit superior performance, showcasing the difficulty of learning the joint quantity and term-based ranking for neural models without a change in architecture or training objective.

We concluded Chapter 5 by looking at the impact of quantity-centric ranking in a RAG pipeline, to answer quantity-centric questions. Despite the quantity-aware models demonstrating better factual correctness for such questions, the generation module of the RAG pipeline introduces a bottleneck for quantity understanding, leaving room for future work in this direction.

6.2 Outlook

Given the diversity of potential applications of quantity-centric ranking and the fact that this area of research is largely unexplored, several open questions warrant further investigation. We identify two limitations of the current models as a basis for future work, namely, *expansion of context beyond sentences* and *considering bounds and valid ranges for quantities in a sentence*. Additionally, we highlight two potential directions for further study and enhancement of quantity-aware retrieval, namely, *quantity-aware loss functions and architectural changes* and *the addition of temporal dimension to search*. Finally, we discuss *quantity-aware generators in RAG pipelines* and the potential for further work in that direction.

Expansion of context beyond sentences: Both the CQE framework and the proposed retrieval architectures in this thesis operate at the sentence level. This deliberate decision was made to preserve the relationships between quantities and concepts during detection and retrieval. As the context window grows longer and more quantities and concepts are involved, this task becomes increasingly challenging. Moreover, with larger window sizes concept detection becomes more difficult, and co-reference resolution becomes more vital to ensure consistency in concept detection. However, condensing the context into a single sentence also increases the risk of errors and the chance of overlooking critical information. In our view, one of the next steps with a focus on the technical aspects of the model is increasing the context size of extraction and retrieval models to go beyond sentence level and include larger chunks of data.

Considering bounds and valid ranges for quantities in a sentence: As mentioned in Chapter 5, our proposed retrieval models assume a simplified setting in a document collection, where the bound-based conditions, where a value falls into a specific range, in the retrieval units are ignored. For example, in "The iPhone XR is priced at under $745 on Amazon", the quantity index in the disjoint model contains the unit "dollar" and a singular value, "745", rather than the entire range of valid values. Designing an index structure that takes ranges of values into account and scoring

functions operating on bounds instead of single values could be useful.
Certain sentences contain speculations or forecasts regarding the future and do not report exact and correct values. For instance, the sentence "S&P 500 is forecasted to drop 1% by end of the year.", presents a speculative prediction of what may occur rather than an actual decline. Considering such cases is another valuable addition to the model

Quantity-aware loss functions and architectural changes: Considering the inferior performance of the joint models in comparison to the disjoint rankers, one obvious area of further study is to inspect more closely the architecture and the objective functions of joint rankers. In particular, it would be interesting to investigate the effect of numerical language models, described in Chapter 3, as base models for retrieval architectures. Since these models have undergone task-specific pre-training for understanding numerical scales, they should be better equipped for quantity comparison and answering quantity-centric queries. Nonetheless, the challenge of fitting such models to general-purpose retrieval remains open, as numerical language modeling tasks have different objectives, e.g., predicting scales rather than the next word. These objective functions need to be altered to account for the representation of quantities and textual elements in an equal manner. Moreover, designing specific fine-tuning objective functions for retrieval is also crucial. Such an objective should account for both numerical proximity and textual similarity. A model trained in this manner can bring the findings on quantity representation learning models into the retrieval settings.

Addition of temporal dimension to search: Temporal information represents a specific type of quantitative data that we have overlooked in this thesis. Nonetheless, temporality remains a crucial aspect of documents, which affects other quantitative information in the text. The quantities associated with concepts are not always static, for instance, while the height of the Eiffel Tower remains constant, the market share value of a stock fluctuates over time. A quantity-aware model equipped to capture temporal dynamics and adept at modeling trends and fluctuations in values can contain more precise and nuanced information from the text. Furthermore, it is rare for a user to conduct a search on a static collection. Integrating the temporal dimension into the retrieval model enables users to more easily locate the information they seek. For example, answering queries such as "How did Apple's revenue change over the past 10 years" or "Did the Bitcoin price in 2022 surpass the peak in 2021?", requires not only quantity understanding but also a search in the temporal dimension. To answer the query "How did the Apple revenue change over the past 10 years", the system should be able to detect the Apple revenue for the past 10 years from the day of the query and provide a summary. Moreover, for the query "Did the Bitcoin price in 2022 surpass the peak in 2021?", not only should the system find the peak price of Bitcoin in 2021 and 2022, but it should also compare them, in order to figure out if one surpasses the other.

Quantity-aware generator in a RAG pipeline: A noteworthy observation from the RAG-based evaluation of our models is the frequent occurrence of flawed generation outputs, even when the generator language model was provided with the correct context. The next step in this area entails exploring avenues to enhance the reasoning ability of the generator concerning quantities in the provided context. Similar to dedicated models for numerical reasoning in questions systems (discussed in Chapter 3), one could either transform contextual chunks into suitable representations showcasing the intrinsic relationships among quantities in the text, or instruct the generator to prioritize quantity relations further, either through task-specific fine-tuning or via in-context learning. For example, the relation between the quantities in the retrieved context could be captured in a graph structure, where nodes are the quantities and the edges show the relationship between the values, *greater than*, *less than*, or *equal*. The graph can be an additional input to the language model to facilitate quantity understanding and the relation between the quantities in a given context.

Acronyms

ANN	Approximate Nearest Neighbour
AP	Average Precision
BCN	Binary Cross Entropy
BERT	Bidirectional Encoder Representations from Transformers
BIR	Binary Relevance Retrieval
BM25	Best Match 25
CBOW	Continuous Bag of Words
CNN	Convolutional Neural Network
ColBERT	Contextualized Late Interaction over BERT
CQE	Comprehensive Quantity Extractor
DCG	Discounted Cumulative Gain
DeepCT	Deep Contextualized Term Weighting
DPR	Dense Passage Retrieval
ELMo	Embedding from Language Models
Faiss	Facebook AI Similarity Search
FFNN	Feed-forward Neural Network
GD	Gradient Descent
GeMM	Generative Masked Measurement model
GloVe	Global Vector embeddings
GMAP	Geometric Mean Average Precision
Grbd	Gorbid-quantities
GRU	Gated Recurrent Unit
HNSW	Hierarchical Navigable Small World
IllQ	Illinois Quantifier
IR	Information Retrieval
LSTM	Long Short Term Memory
MAP	Mean Average Precision
MLM	Masked Language Modeling
MMP	Masked Measurement Prediction
MRR	Mean Reciprocal Rank
NDCG	Normalized Discounted Cumulative Gain
Neural IR	Neural Information Retrieval
NLI	Natural Language Inference
NLL	Negative Log-Likelihood
NLP	Natural Language Processing
NSP	Next Sentence Prediction
NSW	Navigable Small World Graph
NumGNN	Numerical Graph Neural Network
OOV	Out Of Vocabulary

P	Precision
POS	Part of Speech
Q3	Quantulum3
QA	Questions Answering
QBM25	Quantity-aware BM25
QColBERT	Quantity-aware ColBERT
Qfact	Quantity Fact
QSPLADE	Quantity-aware SPLADE
R	Recall
R-Txt	Recognizers-Text
RAG	Retrieval Augmented Generation
RAGAS	Retrieval Augmented Generation Assessment
RBP	Ranked Biased Precision
RLHF	Reinforcement Learning from Human Feedback
RNN	Recurrent Neural Network
RR	Reciprocal Rank
SBERT	Sentence BERT
SG	Skip Gram
SPLADE	SParse Lexical AnD Expansion
SPTAG	Space Partition Tree And Graph
TF-IDF	Term Frequency Inverse Document Frequency

Glossary

$t^{(n)}$	nth token in a sequence
cq	A contextualized quantity
d_t	Bernoulli random variable if t is present in d
ch	Change in quantity value
cn	Concept of a quantity
α	Control parameter for quantity scoring
$e^{(n)}$	Hidden layer of a RNN at timestep n and layer j
fn	False negative
fp	False positive
x_t	Frequency of term t in query x
$\tau, \tau^{'}$	Function to create representation of q and d
G	Gating parameter
$h_j^{(n)}$	Hidden layer of a RNN at timestep n and layer j
idf	Inverse document frequency
dl	Length of a document
K	Matrix of key values in attention mechanism
Q_u	Matrix of query values in attention mechanism
$\mathcal{V}$	Matrix of values in attention mechanism
df_t	Number of documents containing the term t
ef	Number of neighbors in ANN
c	Numerical condition
$\Phi_=$	Numerical scoring function for equal
$\Phi_>$	Numerical scoring function for larger than
$\Phi_{[\,]}$	Numerical scoring function for ranges
$\Phi_<$	Numerical scoring function for smaller than
θ	Parameters of a neural network
PE	Positional Encoding
g_t	Probability that t is present in d given that d is not relevant
p_t	Probability that t is present in d given that d is relevant
X	Query consisting of terms and quantities
r	Relevance
s_-	Sentences from the negative examples to the query
s_o	Sentences from the original sampling strategy
s_+	Sentences from the positive examples to the query
s_f	Sentences of final set of training examples for a query
s_u	Sentences with unit permutation
s_v	Sentences with value permutation
CQ	Set of contextualized quantities
Cu	Set of currencies

Np	Set of noun-based units
V	Set of numerical values
Q	Set of quantities
Si	Set of scientific units
U	Set of units
σ	Sigmoid function
sim	Similarity function
v	Single numerical value
q	Single quantity
x	Single query
t	Single term or token
u	Single unit
$tf_{t,d}$	Term frequency of term t in document d
ω	The weight with a term in a vector space model
tn	True negative
tp	True positive
v_l	Value of the lower bound
v_u	Value of the upper bound
f	Weight for FLOPs regularization
$Vocab$	All the terms in the vocabulary
A	Attention weight matrix
a	Attention weight
b	Bias term
el	Eliteness
e	Embedding vector
E	Embedding weight matrix
h_d	Hidden state of a decoder
h_e	Hidden state of an encoder
I	Importance vector
k	Key vector in attention mechanism
L	Loss function
J	Objective function
$QScore$	Quantity scoring function
q_u	Query vector in attention mechanism
T	Sequence of terms
D	Set of documents
sim_c	Similarity based on numerical condition
d	Single document
ν	Value vector in attention mechanism
W	Weight matrix

Bibliography

Rakesh Agrawal and Ramakrishnan Srikant. Searching with Numbers. *IEEE Trans. Knowl. Data Eng.*, 15(4):855–870, 2003. URL https://doi.org/10.1109/TKDE.2003.1209004.

Satya Almasian, Andreas Spitz, and Michael Gertz. Word Embeddings for Entity-Annotated Texts. In *Advances in Information Retrieval - 41st European Conference on IR Research, ECIR 2019, Cologne, Germany, April 14-18, 2019, Proceedings, Part I*, volume 11437 of *Lecture Notes in Computer Science*, pages 307–322. Springer, 2019. URL https://doi.org/10.1007/978-3-030-15712-8_20.

Satya Almasian, Dennis Aumiller, and Michael Gertz. Bert Got a Date: Introducing Transformers to Temporal Tagging. *arXiv preprint arXiv:2109.14927*, 2021.

Satya Almasian, Dennis Aumiller, and Michael Gertz. Time for some German? Pre-Training a Transformer-based Temporal Tagger for German. In *Proceedings of Text2Story - Fifth Workshop on Narrative Extraction From Texts held in conjunction with the 44th European Conference on Information Retrieval (ECIR 2022), Stavanger, Norway, April 10, 2022*, volume 3117 of *CEUR Workshop Proceedings*, pages 83–90. CEUR-WS.org, 2022a. URL https://ceur-ws.org/Vol-3117/paper9.pdf.

Satya Almasian, Milena Bruseva, and Michael Gertz. QFinder: A Framework for Quantity-centric Ranking. In *SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022*, pages 3272–3277. ACM, 2022b. URL https://doi.org/10.1145/3477495.3531672.

Satya Almasian, Vivian Kazakova, Philip Göldner, and Michael Gertz. CQE: A Comprehensive Quantity Extractor. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 12845–12859. Association for Computational Linguistics, 2023. URL https://aclanthology.org/2023.emnlp-main.793.

Satya Almasian, Vivian Kazakova, and Michael Gertz. Numbers Matter! Bringing Quantity-awareness to Retrieval Systems. Under Review, 2024a.

Satya Almasian, Alexander Kosnac, and Michael Gertz. QuantPlorer: Exploration of Quantities in Text. In *Advances in Information Retrieval - 46th European Conference on Information Retrieval, ECIR 2024, Glasgow, UK, March 24-28, 2024, Proceedings, Part V*, volume 14612 of *Lecture Notes in Computer Science*, pages 171–176. Springer, 2024b. URL https://doi.org/10.1007/978-3-031-56069-9_13.

Omar Alonso and Thibault Sellam. Quantitative Information Extraction From Social Data. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, Ann Arbor, MI, USA, July 08-12, 2018*, pages 1005–1008. ACM, 2018. URL https://doi.org/10.1145/3209978.3210133.

Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D. Manning. Leveraging Linguistic Structure For Open Domain Information Extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 344–354. The Association for Computer Linguistics, 2015. URL https://doi.org/10.3115/v1/p15-1034.

Dogu Araci. FinBERT: Financial Sentiment Analysis with Pre-trained Language Models. *CoRR*, abs/1908.10063, 2019. URL http://arxiv.org/abs/1908.10063.

Dennis Aumiller, Satya Almasian, Philip Hausner, and Michael Gertz. UniHD@CL-SciSumm 2020: Citation Extraction as Search. In *Proceedings of the First Workshop on Scholarly Document Processing*, pages 261–269, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.sdp-1.29. URL https://aclanthology.org/2020.sdp-1.29.

Dennis Aumiller, Satya Almasian, Sebastian Lackner, and Michael Gertz. Structural Text Segmentation of Legal Documents. In *ICAIL '21: Eighteenth International Conference for Artificial Intelligence and Law, São Paulo Brazil, June 21 - 25, 2021*, pages 2–11. ACM, 2021. URL https://doi.org/10.1145/3462757.3466085.

Dennis Aumiller, Satya Almasian, David Pohl, and Michael Gertz. Online DATEing: A Web Interface for Temporal Annotations. In *SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022*, pages 3289–3294. ACM, 2022. URL https://doi.org/10.1145/3477495.3531670.

Martin Aumüller, Erik Bernhardsson, and Alexander John Faithfull. ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms. In *Similarity Search and Applications - 10th International Conference, SISAP 2017, Munich, Germany, October 4-6, 2017, Proceedings*, volume 10609 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2017. URL https://doi.org/10.1007/978-3-319-68474-1_3.

Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *CoRR*, abs/1607.06450, 2016. URL http://arxiv.org/abs/1607.06450.

Ricardo Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval - The Concepts and Technology Behind Search, Second Edition*. Pearson Education Ltd., Harlow, England, 2011. ISBN 978-0-321-41691-9. URL http://www.mir2ed.org/.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.0473.

Yang Bai, Xiaoguang Li, Gang Wang, Chaoliang Zhang, Lifeng Shang, Jun Xu, Zhaowei Wang, Fangshan Wang, and Qun Liu. SparTerm: Learning Term-based Sparse Representation for Fast Text Retrieval. *CoRR*, abs/2010.00768, 2020. URL https://arxiv.org/abs/2010.00768.

Somnath Banerjee, Soumen Chakrabarti, and Ganesh Ramakrishnan. Learning to Rank for Quantity Consensus Queries. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009, Boston, MA, USA, July 19-23, 2009*, pages 243–250. ACM, 2009. URL https://doi.org/10.1145/1571941.1571985.

Jeffrey S. Beis and David G. Lowe. Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces. In *1997 Conference on Computer Vision and Pattern Recognition (CVPR'97), June 17-19, 1997, San Juan, Puerto Rico*, pages 1000–1006. IEEE Computer Society, 1997. URL https://doi.org/10.1109/CVPR.1997.609451.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A Neural Probabilistic Language Model. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pages 932–938. MIT Press, 2000. URL https://proceedings.neurips.cc/paper/2000/hash/728f206c2a01bf572b5940d7d9a8fa4c-Abstract.html.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3:1137–1155, 2003. URL http://jmlr.org/papers/v3/bengio03a.html.

Jon Louis Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, 18(9):509–517, 1975. URL https://doi.org/10.1145/361002.361007.

Taylor Berg-Kirkpatrick and Daniel Spokoyny. An Empirical Investigation of Contextualized Number Prediction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 4754–4764. Association for Computational Linguistics, 2020. URL https://doi.org/10.18653/v1/2020.emnlp-main.385.

Soumia Lilia Berrahou, Patrice Buche, Juliette Dibie, and Mathieu Roche. Xart: Discovery of Correlated Arguments of n-ary Relations in Text. *Expert Systems with Applications*, 73:115–124, 2017. URL https://doi.org/10.1016/j.eswa.2016.12.028.

Michael W. Berry, Zlatko Drmac, and Elizabeth R. Jessup. Matrices, Vector Spaces, and Information Retrieval. *SIAM Review*, 41(2):335–362, 1999. URL https://doi.org/10.1137/S0036144598347035.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. URL https://doi.org/10.1162/tacl_a_00051.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, and Bogdan Damoc et al. Improving Language Models by Retrieving from Trillions of Tokens. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 2206–2240. PMLR, 2022. URL https://proceedings.mlr.press/v162/borgeaud22a.html.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and Sandhini Agarwal et al. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

C Buckley and Ellen Voorhees. *Retrieval System Evaluation*. TREC: Experiment and Evaluation in Information Retrieval, September 2005.

Michael Burrows. *Object-Oriented Interface for an Index*. US patent 5809502, 1996.

Angel X. Chang and Christopher D. Manning. SUTime: Evaluation in TempEval-3. In *Proceedings of the 7th International Workshop on Semantic Evaluation, SemEval@NAACL-HLT 2013, Atlanta, Georgia, USA, June 14-15, 2013*, pages 78–82. The Association for Computer Linguistics, 2013. URL https://aclanthology.org/S13-2013/.

Chung-Chi Chen, Hen-Hsen Huang, Hiroya Takamura, and Hsin-Hsi Chen. Numeracy-600K: Learning Numeracy for Detecting Exaggerated Information in Market Comments. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 6307–6313. Association for Computational Linguistics, 2019a. URL https://doi.org/10.18653/v1/p19-1635.

Chung-Chi Chen, Hen-Hsen Huang, and Hsin-Hsi Chen. NumClaim: Investor's Fine-grained Claim Detection. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, pages 1973–1976. ACM, 2020a. URL https://doi.org/10.1145/3340531.3412100.

Chung-Chi Chen, Hen-Hsen Huang, and Hsin-Hsi Chen. NQuAD: 70, 000+ Questions for Machine Comprehension of the Numerals in Text. In *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 2925–2929. ACM, 2021. URL https://doi.org/10.1145/3459637.3482155.

Kunlong Chen, Weidi Xu, Xingyi Cheng, Zou Xiaochuan, Yuyu Zhang, Le Song, Taifeng Wang, Yuan Qi, and Wei Chu. Question Directed Graph Attention Network for Numerical Reasoning over Text. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 6759–6768. Association for Computational Linguistics, 2020b. URL https://doi.org/10.18653/v1/2020.emnlp-main.549.

Qi Chen, Haidong Wang, Mingqin Li, Gang Ren, Scarlett Li, Jeffery Zhu, Jason Li, Chuanjie Liu, Lintao Zhang, and Jingdong Wang. *SPTAG: A Library for Fast Approximate Nearest Neighbor Search*. GitHub. https://github. com/Microsoft/SPTAG, 2018.

Ruey-Cheng Chen, Luke Gallagher, Roi Blanco, and J. Shane Culpepper. Efficient Cost-Aware Cascade Ranking in Multi-Stage Retrieval. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, pages 445–454. ACM, 2017. URL https://doi.org/10.1145/3077136.3080819.

Sanxing Chen, Guoxin Wang, and Börje Karlsson. Exploring Word Representations on Time Expression Recognition. *Microsoft Research Asia, Technology Representatives*, June 2019b.

Xinyun Chen, Chen Liang, Adams Wei Yu, Denny Zhou, Dawn Song, and Quoc V. Le. Neural Symbolic Reader: Scalable Integration of Distributed and Symbolic Representations for Reading Comprehension. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020c. URL https://openreview.net/forum?id=ryxjnREFwH.

Colin Cherry and Hongyu Guo. The Unreasonable Effectiveness of Word Representations for Twitter Named Entity Recognition. In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies,*

Denver, Colorado, USA, May 31 - June 5, 2015, pages 735–745. The Association for Computational Linguistics, 2015. URL https://doi.org/10.3115/v1/n15-1075.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. In *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*, pages 103–111. Association for Computational Linguistics, 2014a. doi: 10.3115/v1/W14-4012. URL https://aclanthology.org/W14-4012/.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL, 2014b. URL https://doi.org/10.3115/v1/d14-1179.

Keunwoo Choi, György Fazekas, Mark B. Sandler, and Kyunghyun Cho. Convolutional Recurrent Neural Networks for Music Classification. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017, New Orleans, LA, USA, March 5-9, 2017*, pages 2392–2396. IEEE, 2017. URL https://doi.org/10.1109/ICASSP.2017.7952585.

Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep Reinforcement Learning from Human Preferences. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4299–4307, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/d5e2c0adad503c91f91df240d0cd4e49-Abstract.html.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=r1xMH1BtvB.

Stéphane Clinchant and Florent Perronnin. Aggregating Continuous Word Embeddings for Information Retrieval. In *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality, CVSM@ACL 2013, Sofia, Bulgaria, August 9, 2013*, pages 100–109. Association for Computational Linguistics, 2013. URL https://aclanthology.org/W13-3212/.

Jacob Cohen. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement, Sage Publications Sage CA: Thousand Oaks, CA*, 20(1):37–46, 1960.

Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009, Boston, MA, USA, July 19-23, 2009*, pages 758–759. ACM, 2009. URL https://doi.org/10.1145/1571941.1572114.

W. Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Information Retrieval in Practice*. Pearson Education, 2009. ISBN 978-0-13-136489-9. URL http://www.search-engines-book.com/.

Hamish Cunningham, Valentin Tablan, Angus Roberts, and Kalina Bontcheva. Getting More Out of Biomedical Documents with GATE's Full Lifecycle Open Source Text Analytics. *PLoS Comput. Biol.*, 9(2), 2013. URL https://doi.org/10.1371/journal.pcbi.1002854.

Andrew M. Dai and Quoc V. Le. Semi-supervised Sequence Learning. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 3079–3087, 2015. URL https://proceedings.neurips.cc/paper/2015/hash/7137debd45ae4d0ab9aa953017286b20-Abstract.html.

Zhuyun Dai and Jamie Callan. Context-Aware Sentence/Passage Term Importance Estimation For First Stage Retrieval. *CoRR*, abs/1910.10687, 2019. URL http://arxiv.org/abs/1910.10687.

Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive Hashing Scheme Based on p-stable Distributions. In *Proceedings of the 20th ACM Symposium on Computational Geometry, Brooklyn, New York, USA, June 8-11, 2004*, pages 253–262. ACM, 2004. URL https://doi.org/10.1145/997817.997857.

Hervé Déjean, Stéphane Clinchant, and Thibault Formal. A Thorough Comparison of Cross-Encoders and LLMs for Reranking SPLADE. *CoRR*, abs/2403.10407, 2024. URL https://doi.org/10.48550/arXiv.2403.10407.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019a. URL https://doi.org/10.18653/v1/n19-1423.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019b. URL https://doi.org/10.18653/v1/n19-1423.

Iddo Drori, Sunny Tran, Roman Wang, Newman Cheng, Kevin Liu, Leonard Tang, Elizabeth Ke, Nikhil Singh, Taylor L. Patti, Jayson Lynch, Avi Shporer, Nakul Verma, Eugene Wu, and Gilbert Strang. A Neural Network Solves and Generates Mathematics Problems by Program Synthesis: Calculus, Differential Equations, Linear Algebra, and More. *CoRR*, abs/2112.15594, 2021. URL https://arxiv.org/abs/2112.15594.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A Reading Comprehension Benchmark Requiring Discrete Reasoning Over Paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 2368–2378. Association for Computational Linguistics, 2019. URL https://doi.org/10.18653/v1/n19-1246.

Abhijeet Dubey, Lakshya Kumar, Arpan Somani, Aditya Joshi, and Pushpak Bhattacharyya. "When Numbers Matter!": Detecting Sarcasm in Numerical Portions of Text. In *Proceedings of the Tenth*

Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, WASSA@NAACL-HLT 2019, Minneapolis, USA, June 6, 2019, pages 72–80. Association for Computational Linguistics, 2019. URL https://doi.org/10.18653/v1/w19-1309.

Jacob Eisenstein. *Introduction to Natural Language Processing*. Adaptive Computation and Machine Learning Series. MIT Press, 2019. ISBN 978-0-262-04284-0.

Yanai Elazar, Abhijit Mahabal, Deepak Ramachandran, Tania Bedrax-Weiss, and Dan Roth. How Large Are Lions? Inducing Distributions over Quantitative Attributes. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 3973–3983. Association for Computational Linguistics, 2019. URL https://doi.org/10.18653/v1/p19-1388.

Shahul ES, Jithin James, Luis Espinosa Anke, and Steven Schockaert. RAGAs: Automated Evaluation of Retrieval Augmented Generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2024 - System Demonstrations, St. Julians, Malta, March 17-22, 2024*, pages 150–158. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.eacl-demo.16.

Jing Fan, Dennis Aumiller, and Michael Gertz. Evaluating Factual Consistency of Texts with Semantic Role Labeling. In *Proceedings of the The 12th Joint Conference on Lexical and Computational Semantics, *SEM@ACL 2023, Toronto, Canada, July 13-14, 2023*, pages 89–100. Association for Computational Linguistics, 2023. URL https://doi.org/10.18653/v1/2023.starsem-1.9.

J. R. Firth. *Papers in Linguistics, 1934-1951*. Oxford University Press, 1957.

Marcus Fontoura, Ronny Lempel, Runping Qi, and Jason Y. Zien. Inverted Index Support for Numeric Search. *Internet Math.*, 3(2):153–185, 2007. URL https://doi.org/10.1080/15427951.2006.10129119.

Luca Foppiano, Laurent Romary, Masashi Ishii, and Mikiko Tanifuji. Automatic Identification and Normalisation of Physical Measurements in Scientific Literature. In *Proceedings of the ACM Symposium on Document Engineering 2019, Berlin, Germany, September 23-26, 2019*, pages 24:1–24:4. ACM, 2019. URL https://doi.org/10.1145/3342558.3345411.

Kenneth D. Forbus. Qualitative Process Theory: Twelve Years After. *Artificial Intelligence*, 59(1-2): 115–123, 1993. URL https://doi.org/10.1016/0004-3702(93)90177-D.

Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. SPLADE v2: Sparse Lexical and Expansion Model for Information Retrieval. *CoRR*, abs/2109.10086, 2021a. URL https://arxiv.org/abs/2109.10086.

Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, pages 2288–2292. ACM, 2021b. URL https://doi.org/10.1145/3404835.3463098.

Norbert Fuhr. Some Common Mistakes In IR Evaluation, And How They Can Be Avoided. *SIGIR Forum*, 51(3):32–41, 2017. URL https://doi.org/10.1145/3190580.3190586.

Philip Gage. A New Algorithm for Data Compression. *C Users Journal, McPherson, KS: R & D Publications*, 12(2):23–38, 1994.

Luyu Gao, Yunyi Zhang, Jiawei Han, and Jamie Callan. Scaling Deep Contrastive Learning Batch Size under Memory Limited Setup. In *Proceedings of the 6th Workshop on Representation Learning for NLP, RepL4NLP@ACL-IJCNLP 2021, Online, August 6, 2021*, pages 316–321. Association for Computational Linguistics, 2021. URL https://doi.org/10.18653/v1/2021.repl4nlp-1.31.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise Zero-Shot Dense Retrieval without Relevance Labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 1762–1777. Association for Computational Linguistics, 2023a. URL https://doi.org/10.18653/v1/2023.acl-long.99.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. Retrieval-Augmented Generation for Large Language Models: A Survey. *CoRR*, abs/2312.10997, 2023b. URL https://doi.org/10.48550/arXiv.2312.10997.

Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized Product Quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(4):744–755, 2014. URL https://doi.org/10.1109/TPAMI.2013.240.

Felix A. Gers, Jürgen Schmidhuber, and Fred A. Cummins. Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000. URL https://doi.org/10.1162/089976600300015015.

Mor Geva, Ankit Gupta, and Jonathan Berant. Injecting Numerical Reasoning Skills into Language Models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 946–958. Association for Computational Linguistics, 2020. URL https://doi.org/10.18653/v1/2020.acl-main.89.

Daniel Gillick, Alessandro Presta, and Gaurav Singh Tomar. End-to-End Retrieval in Continuous Space. *CoRR*, abs/1811.08008, 2018. URL http://arxiv.org/abs/1811.08008.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. ToRA: A Tool-Integrated Reasoning Agent for Mathematical Problem Solving. *CoRR*, abs/2309.17452, 2023. URL https://doi.org/10.48550/arXiv.2309.17452.

Laura A. Granka, Thorsten Joachims, and Geri Gay. Eye-tracking analysis of user behavior in WWW search. In *SIGIR 2004: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Sheffield, UK, July 25-29, 2004*, pages 478–479. ACM, 2004. URL https://doi.org/10.1145/1008992.1009079.

Alex Graves. Generating Sequences With Recurrent Neural Networks. *CoRR*, abs/1308.0850, 2013. URL http://arxiv.org/abs/1308.0850.

Jiafeng Guo, Yinqiong Cai, Yixing Fan, Fei Sun, Ruqing Zhang, and Xueqi Cheng. Semantic Models for the First-Stage Retrieval: A Comprehensive Review. *ACM Transactions on Information Systems*, 40(4):66:1–66:42, 2022. URL https://doi.org/10.1145/3486250.

Nitish Gupta, Kevin Lin, Dan Roth, Sameer Singh, and Matt Gardner. Neural Module Networks for Reasoning over Text. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SygWvAVFPr.

Suchin Gururangan, Ana Marasovic, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don't Stop Pretraining: Adapt Language Models to Domains and Tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 8342–8360. Association for Computational Linguistics, 2020. URL https://doi.org/10.18653/v1/2020.acl-main.740.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. REALM: Retrieval-Augmented Language Model Pre-Training. *CoRR*, abs/2002.08909, 2020. URL https://arxiv.org/abs/2002.08909.

Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, Wentao Han, Minlie Huang, Qin Jin, Yanyan Lan, Yang Liu, Zhiyuan Liu, Zhiwu Lu, Xipeng Qiu, Ruihua Song, Jie Tang, Ji-Rong Wen, Jinhui Yuan, Wayne Xin Zhao, and Jun Zhu. Pre-trained models: Past, present and future. *AI Open*, 2:225–250, 2021. URL https://doi.org/10.1016/j.aiopen.2021.08.002.

Stephen P. Harter. A probabilistic approach to automatic keyword indexing. Part I. On the Distribution of Specialty Words in a Technical Literature. *Journal of the Association for Information Science and Technology*, 26(4):197–206, 1975. URL https://doi.org/10.1002/asi.4630260402.

Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 1923–1933. Association for Computational Linguistics, 2017. URL https://doi.org/10.18653/v1/d17-1206.

Philip Hausner, Dennis Aumiller, and Michael Gertz. TiCCo: Time-Centric Content Exploration. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, pages 3413–3416. ACM, 2020. URL https://doi.org/10.1145/3340531.3417432.

Hangfeng He, Hongming Zhang, and Dan Roth. Rethinking with Retrieval: Faithful Large Language Model Inference. *CoRR*, abs/2301.00303, 2023. URL https://doi.org/10.48550/arXiv.2301.00303.

Matthew L. Henderson, Rami Al-Rfou, Brian Strope, Yun-Hsuan Sung, László Lukács, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. Efficient Natural Language Response Suggestion for Smart Reply. *CoRR*, abs/1705.00652, 2017. URL http://arxiv.org/abs/1705.00652.

Vinh Thinh Ho, Yusra Ibrahim, Koninika Pal, Klaus Berberich, and Gerhard Weikum. Qsearch: Answering Quantity Queries from Text. In *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part I*, volume 11778 of *Lecture Notes in Computer Science*, pages 237–257. Springer, 2019. URL https://doi.org/10.1007/978-3-030-30793-6_14.

Vinh Thinh Ho, Koninika Pal, Niko Kleer, Klaus Berberich, and Gerhard Weikum. Entities with Quantities: Extraction, Search, and Ranking. In *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining*, pages 833–836. ACM, 2020. URL https://doi.org/10.1145/3336191.3371860.

Vinh Thinh Ho, Koninika Pal, Simon Razniewski, Klaus Berberich, and Gerhard Weikum. Extracting Contextualized Quantity Facts from Web Tables. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 4033–4042. ACM / IW3C2, 2021a. URL https://doi.org/10.1145/3442381.3450072.

Vinh Thinh Ho, Koninika Pal, and Gerhard Weikum. QuTE: Answering Quantity Queries from Web Tables. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 2740–2744. ACM, 2021b. URL https://doi.org/10.1145/3448016.3452763.

Vinh Thinh Ho, Daria Stepanova, Dragan Milchevski, Jannik Strötgen, and Gerhard Weikum. Enhancing Knowledge Bases with Quantity Facts. In *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, pages 893–901. ACM, 2022. URL https://doi.org/10.1145/3485447.3511932.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8): 1735–1780, 1997. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen Fürstenau, Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. Robust Disambiguation of Named Entities in Text. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27-31 July 2011, John McIntyre Conference Centre, Edinburgh, UK, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 782–792. ACL, 2011. URL https://aclanthology.org/D11-1072/.

Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke Zettlemoyer, and Daniel S. Weld. Knowledge-Based Weak Supervision for Information Extraction of Overlapping Relations. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 541–550. The Association for Computer Linguistics, 2011. URL https://aclanthology.org/P11-1055/.

Sebastian Hofstätter, Omar Khattab, Sophia Althammer, Mete Sertkan, and Allan Hanbury. Introducing Neural Bag of Whole-Words with ColBERTer: Contextualized Late Interactions using Enhanced Reduction. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022*, pages 737–747. ACM, 2022. URL https://doi.org/10.1145/3511808.3557367.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. Learning to Solve Arithmetic Word Problems with Verb Categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 523–533. ACL, 2014. URL https://doi.org/10.3115/v1/d14-1058.

Wenhao Huang, Zijia Lin, Chris McConnell, and Börje F. Karlsson. *Recognizers-Text: Recognition and Resolution of Numbers, Units, and Date/time Entities Expressed Across Multiple Languages*. Zenodo, July 2017. URL https://doi.org/10.5281/zenodo.6860598.

Yusra Ibrahim and Gerhard Weikum. ExQuisiTe: Explaining Quantities in Text. In *The World Wide Web Conference, WWW*, pages 3541–3544. ACM, 2019. URL https://doi.org/10.1145/3308558.3314134.

Yusra Ibrahim, Mirek Riedewald, Gerhard Weikum, and Demetrios Zeinalipour-Yazti. Bridging Quantities in Tables and Text. In *35th IEEE International Conference on Data Engineering, ICDE*, pages 1010–1021. IEEE, 2019. URL https://doi.org/10.1109/ICDE.2019.00094.

Piotr Indyk and Rajeev Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 604–613. ACM, 1998. URL https://doi.org/10.1145/276698.276876.

Gautier Izacard and Edouard Grave. Distilling Knowledge from Reader to Retriever for Question Answering. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=NTEz-6wysdb.

Kyoungrok Jang, Junmo Kang, Giwon Hong, Sung-Hyon Myaeng, Joohee Park, Taewon Yoon, and Hee-Cheol Seo. Ultra-High Dimensional Sparse Representations with Binarization for Efficient Text Retrieval. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 1016–1029. Association for Computational Linguistics, 2021. URL https://doi.org/10.18653/v1/2021.emnlp-main.78.

Kalervo Järvelin and Jaana Kekäläinen. Cumulated Gain-based Evaluation of IR Techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002. URL http://doi.acm.org/10.1145/582415.582418.

Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011. URL https://doi.org/10.1109/TPAMI.2010.57.

Chengyue Jiang, Zhonglin Nian, Kaihao Guo, Shanbo Chu, Yinggong Zhao, Libin Shen, and Kewei Tu. Learning Numeral Embedding. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 2586–2599. Association for Computational Linguistics, 2020. URL https://doi.org/10.18653/v1/2020.findings-emnlp.235.

Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active Retrieval Augmented Generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 7969–7992. Association for Computational Linguistics, 2023. URL https://aclanthology.org/2023.emnlp-main.495.

Zhihua Jin, Xin Jiang, Xingbo Wang, Qun Liu, Yong Wang, Xiaozhe Ren, and Huamin Qu. NumGPT: Improving Numeracy Ability of Generative Pre-trained Models. *CoRR*, abs/2109.03137, 2021. URL https://arxiv.org/abs/2109.03137.

Thorsten Joachims. Optimizing Search Engines Using Clickthrough Data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*, pages 133–142. ACM, 2002. URL https://doi.org/10.1145/775047.775067.

Ari K. Tuchman John K. Stockton. *System and Methods for Units-Based Numeric Information Retrieval*. US patent 20100332511, 2009.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2021. URL https://doi.org/10.1109/TBDATA.2019.2921572.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. SpanBERT: Improving Pre-training by Representing and Predicting Spans. *Transactions of the Association for Computational Linguistics*, 8:64–77, 2020. URL https://doi.org/10.1162/tacl_a_00300.

Dan Jurafsky and James H. Martin. *Speech and Language Processing: an Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, 2nd Edition*. Prentice Hall series in artificial intelligence. Prentice Hall, Pearson Education International, 2009. ISBN 9780135041963. URL https://www.worldcat.org/oclc/315913020.

Nikhil Kandpal, Haikang Deng, Adam Roberts, Eric Wallace, and Colin Raffel. Large Language Models Struggle to Learn Long-Tail Knowledge. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 15696–15707. PMLR, 2023. URL https://proceedings.mlr.press/v202/kandpal23a.html.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 6769–6781. ACL, 2020. URL https://doi.org/10.18653/v1/2020.emnlp-main.550.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through Memorization: Nearest Neighbor Language Models. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=HklBjCEKvH.

Omar Khattab and Matei Zaharia. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48. ACM, 2020. URL https://doi.org/10.1145/3397271.3401075.

Yoon Kim. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751. ACL, 2014. URL https://doi.org/10.3115/v1/d14-1181.

Philipp Koehn. Statistical Significance Tests for Machine Translation Evaluation. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing , EMNLP 2004, A meeting of SIGDAT, a Special Interest Group of the ACL, held in conjunction with ACL 2004, 25-26 July 2004, Barcelona, Spain*, pages 388–395. ACL, 2004. URL https://aclanthology.org/W04-3250/.

Teuvo Kohonen. The Self-organizing Map. *Neurocomputing*, 21(1-3):1–6, 1998. URL https://doi.org/10.1016/S0925-2312(98)00030-7.

Klaus Krippendorff. *Content analysis: An introduction to its methodology*. Sage publications, 2018.

Taku Kudo and John Richardson. SentencePiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018*, pages 66–71. Association for Computational Linguistics, 2018. URL https://doi.org/10.18653/v1/d18-2012.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=H1eA7AEtvS.

Lukas Lange, Anastasiia Iurshina, Heike Adel, and Jannik Strötgen. Adversarial Alignment of Multilingual Models for Extracting Temporal Expressions from Text. In *Proceedings of the 5th Workshop on Representation Learning for NLP, RepL4NLP@ACL 2020, Online, July 9, 2020*, pages 103–109. Association for Computational Linguistics, 2020. URL https://doi.org/10.18653/v1/2020.repl4nlp-1.14.

Egoitz Laparra, Dongfang Xu, and Steven Bethard. From Characters to Time Intervals: New Paradigms for Evaluation and Neural Parsing of Time Normalizations. *Transactions of the Association for Computational Linguistics*, 6:343–356, 2018. URL https://doi.org/10.1162/tacl_a_00025.

Maxime Lefrançois and Antoine Zimmermann. The Unified Code for Units of Measure in RDF: cdt: ucum and other UCUM Datatypes. In *The Semantic Web: ESWC 2018 Satellite Events - ESWC 2018 Satellite Events, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers*, volume 11155 of *Lecture Notes in Computer Science*, pages 196–201. Springer, 2018. URL https://doi.org/10.1007/978-3-319-98192-5_37.

Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

Hang Li. *Learning to Rank for Information Retrieval and Natural Language Processing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2011. URL https://doi.org/10.2200/S00348ED1V01Y201104HLT012.

Hang Li, Tie-Yan Liu, and ChengXiang Zhai. Learning to Rank for Information Retrieval (LR4IR 2008). *SIGIR Forum*, 42(2):76–79, 2008. URL https://doi.org/10.1145/1480506.1480519.

Tongliang Li, Lei Fang, Jian-Guang Lou, Zhoujun Li, and Dongmei Zhang. AnaSearch: Extract, Retrieve and Visualize Structured Results from Unstructured Text for Analytical Queries. In *WSDM '21, The Fourteenth ACM International Conference on Web Search and Data Mining, Virtual Event, Israel, March 8-12, 2021*, pages 906–909. ACM, 2021. URL https://doi.org/10.1145/3437963.3441694.

Zehan Li, Nan Yang, Liang Wang, and Furu Wei. Learning Diverse Document Representations with Deep Query Interactions for Dense Retrieval. *CoRR*, abs/2208.04232, 2022. URL https://doi.org/10.48550/arXiv.2208.04232.

Bill Yuchen Lin, Seyeon Lee, Rahul Khanna, and Xiang Ren. Birds have four legs?! NumerSense: Probing Numerical Commonsense Knowledge of Pre-Trained Language Models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 6862–6868. Association for Computational Linguistics, 2020a. URL https://doi.org/10.18653/v1/2020.emnlp-main.557.

Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. Distilling Dense Representations for Ranking using Tightly-Coupled Teachers. *CoRR*, abs/2010.11386, 2020b. URL https://arxiv.org/abs/2010.11386.

Ling Liu and M. Tamer Özsu, editors. *Encyclopedia of Database Systems, Second Edition*. Springer, 2018. ISBN 978-1-4614-8266-6. URL https://doi.org/10.1007/978-1-4614-8265-9.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the Middle: How Language Models Use Long Contexts. *CoRR*, abs/2307.03172, 2023. URL https://doi.org/10.48550/arXiv.2307.03172.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR*, abs/1907.11692, 2019. URL http://arxiv.org/abs/1907.11692.

Lefteris Loukas, Manos Fergadiotis, Ilias Chalkidis, Eirini Spyropoulou, Prodromos Malakasiotis, Ion Androutsopoulos, and Georgios Paliouras. FiNER: Financial Numeric Entity Recognition for XBRL Tagging. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 4419–4431. Association for Computational Linguistics, 2022. URL https://aclanthology.org/2022.acl-long.303.

Jing Lu, Gustavo Hernández Ábrego, Ji Ma, Jianmo Ni, and Yinfei Yang. Multi-stage Training with Improved Negative Contrast for Neural Passage Retrieval. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 6091–6103. Association for Computational Linguistics, 2021. URL https://doi.org/10.18653/v1/2021.emnlp-main.492.

Pan Lu, Liang Qiu, Wenhao Yu, Sean Welleck, and Kai-Wei Chang. A Survey of Deep Learning for Mathematical Reasoning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*,

pages 14605–14631. Association for Computational Linguistics, 2023. URL https://doi.org/10.18653/v1/2023.acl-long.817.

Aman Madaan, Ashish R. Mittal, Mausam, Ganesh Ramakrishnan, and Sunita Sarawagi. Numerical Relation Extraction with Minimal Supervision. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 2764–2771. AAAI Press, 2016. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12486.

Arun S. Maiya, Dale Visser, and Andrew Wan. Mining Measured Information from Text. In *Proceedings of the 38th International SIGIR Conference on Research and Development in Information Retrieval*, pages 899–902. ACM, 2015. URL https://doi.org/10.1145/2766462.2767789.

Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Scalable Distributed Algorithm for Approximate Nearest Neighbor Search Problem in High Dimensional General Metric Spaces. In *Similarity Search and Applications - 5th International Conference, SISAP 2012, Toronto, ON, Canada, August 9-10, 2012. Proceedings*, volume 7404 of *Lecture Notes in Computer Science*, pages 132–147. Springer, 2012. URL https://doi.org/10.1007/978-3-642-32153-5_10.

Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate Nearest Neighbor Algorithm Based on Navigable Small World Graphs. *Information Systems*, 45:61–68, 2014. URL https://doi.org/10.1016/j.is.2013.10.006.

Yury A. Malkov and Dmitry A. Yashunin. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020. URL https://doi.org/10.1109/TPAMI.2018.2889473.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. ISBN 978-0-521-86571-5. URL https://nlp.stanford.edu/IR-book/pdf/irbookprint.pdf.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, System Demonstrations*, pages 55–60. The Association for Computer Linguistics, 2014. URL https://doi.org/10.3115/v1/p14-5010.

Gary Marcus. The Next Decade in AI: Four Steps Towards Robust Artificial Intelligence. *CoRR*, abs/2002.06177, 2020. URL https://arxiv.org/abs/2002.06177.

Runping Qi Marcus Fontoura, Ronny Lempel and Jason Zien. *Method, System, and Program for Searching Documents for Ranges of Numeric Values*. US patent 20060074962, 2004.

Oren Melamud, Jacob Goldberger, and Ido Dagan. Context2vec: Learning Generic Context Embedding with Bidirectional LSTM. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016*, pages 51–61. ACL, 2016. URL https://doi.org/10.18653/v1/k16-1006.

Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013. URL http://arxiv.org/abs/1301.3781.

George A Miller. *WordNet: An Electronic Lexical Database*. The MIT press, 1998.

Swaroop Mishra, Matthew Finlayson, Pan Lu, Leonard Tang, Sean Welleck, Chitta Baral, Tanmay Rajpurohit, Oyvind Tafjord, Ashish Sabharwal, Peter Clark, and Ashwin Kalyan. LILA: A Unified Benchmark for Mathematical Reasoning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 5807–5832. Association for Computational Linguistics, 2022a. URL https://doi.org/10.18653/v1/2022.emnlp-main.392.

Swaroop Mishra, Arindam Mitra, Neeraj Varshney, Bhavdeep Singh Sachdeva, Peter Clark, Chitta Baral, and Ashwin Kalyan. NumGLUE: A Suite of Fundamental yet Challenging Mathematical Reasoning Tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 3505–3523. Association for Computational Linguistics, 2022b. URL https://doi.org/10.18653/v1/2022.acl-long.246.

Bhaskar Mitra and Nick Craswell. An Introduction to Neural Information Retrieval. *Found. Trends Inf. Retr.*, 13(1):1–126, 2018. URL https://doi.org/10.1561/1500000061.

Moran Mizrahi, Guy Kaplan, Dan Malkin, Rotem Dror, Dafna Shahaf, and Gabriel Stanovsky. State of What Art? A Call for Multi-Prompt LLM Evaluation. *CoRR*, abs/2401.00595, 2024. doi: 10.48550/ARXIV.2401.00595. URL https://doi.org/10.48550/arXiv.2401.00595.

Alistair Moffat and Justin Zobel. Rank-biased Precision for Measurement of Retrieval Effectiveness. *ACM Transactions on Information Systems*, 27(1):2:1–2:27, 2008. URL https://doi.org/10.1145/1416950.1416952.

Aakanksha Naik, Abhilasha Ravichander, Carolyn Penstein Rosé, and Eduard H. Hovy. Exploring Numeracy in Word Embeddings. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 3374–3380. Association for Computational Linguistics, 2019. URL https://doi.org/10.18653/v1/p19-1329.

David Newell and Eite Tiesinga. *The International System of Units (SI), 2019 Edition*. Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, 2019-08-20 2019. doi: https://doi.org/10.6028/NIST.SP.330-2019.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. In *Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS)*, volume 1773 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016. URL https://ceur-ws.org/Vol-1773/CoCoNIPS_2016_paper9.pdf.

GF Nicholson. An Introduction to the Rationalized MKS System of Units. *British Journal of Applied Physics*, 2(7):177, 1951.

Qiang Ning, Ben Zhou, Hao Wu, Haoruo Peng, Chuchu Fan, and Matt Gardner. A Meta-framework for Spatiotemporal Quantity Extraction from Text. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 2736–2749. Association for Computational Linguistics, 2022. URL https://aclanthology.org/2022.acl-long.195.

Rodrigo Nogueira and Kyunghyun Cho. Passage Re-ranking with BERT. *CoRR*, abs/1901.04085, 2019. URL http://arxiv.org/abs/1901.04085.

Rodrigo Nogueira, Jimmy Lin, and AI Epistemic. From doc2query to docTTTTTquery. *Online preprint*, 6, 2019a. URL https://cs.uwaterloo.ca/~jimmylin/publications/Nogueira_Lin_2019_docTTTTTquery-v2.pdf.

Rodrigo Frassetto Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. Document Expansion by Query Prediction. *CoRR*, abs/1904.08375, 2019b. URL http://arxiv.org/abs/1904.08375.

Maxwell I. Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show Your Work: Scratchpads for Intermediate Computation with Language Models. *CoRR*, abs/2112.00114, 2021. URL https://arxiv.org/abs/2112.00114.

Josh Achiam OpenAI, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, and Janko Altenschmidt et al. GPT-4 Technical Report. *CoRR*, abs/2303.08774, 2023. URL https://doi.org/10.48550/arXiv.2303.08774.

Biswajit Paria, Chih-Kuan Yeh, Ian En-Hsu Yen, Ning Xu, Pradeep Ravikumar, and Barnabás Póczos. Minimizing FLOPs to Learn Efficient Sparse Representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SygpC6Ntvr.

Razvan Pascanu, Tomás Mikolov, and Yoshua Bengio. On the Difficulty of Training Recurrent Neural Networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1310–1318. JMLR.org, 2013. URL http://proceedings.mlr.press/v28/pascanu13.html.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543. ACL, 2014. URL https://doi.org/10.3115/v1/d14-1162.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics, 2018. URL https://doi.org/10.18653/v1/n18-1202.

Dominic Petrak, Nafise Sadat Moosavi, and Iryna Gurevych. Arithmetic-Based Pretraining Improving Numeracy of Pretrained Language Models. In *Proceedings of the The 12th Joint Conference on Lexical and Computational Semantics, *SEM@ACL 2023, Toronto, Canada, July 13-14, 2023*, pages 477–493. Association for Computational Linguistics, 2023. URL https://doi.org/10.18653/v1/2023.starsem-1.42.

Alexander Ponomarenko, Yury Malkov, Andrey Logvinov, and Vladimir Krylov. Approximate Nearest Neighbor Search Small World Approach. In *International Conference on Information and Communication Technologies & Applications*, volume 17, 2011.

William W. Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees. *Communications of the ACM*, 33(6):668–676, 1990. URL https://doi.org/10.1145/78973.78977.

Vasin Punyakanok and Dan Roth. The Use of Classifiers in Sequential Inference. In *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pages 995–1001. MIT Press, 2000. URL https://proceedings.neurips.cc/paper/2000/hash/f4a4da9aa7eadfd23c7bdb7cf57b3112-Abstract.html.

Yifan Qiao, Chenyan Xiong, Zhenghao Liu, and Zhiyuan Liu. Understanding the Behaviors of BERT in Ranking. *CoRR*, abs/1904.07531, 2019. URL http://arxiv.org/abs/1904.07531.

Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. RocketQA: An Optimized Training Approach to Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 5835–5847. Association for Computational Linguistics, 2021. URL https://doi.org/10.18653/v1/2021.naacl-main.466.

Zackary Rackauckas. RAG-Fusion: a New Take on Retrieval-Augmented Generation. *CoRR*, abs/2402.03367, 2024. URL https://doi.org/10.48550/arXiv.2402.03367.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. *Improving Language Understanding by Generative Pre-training*. OpenAI, 2018. URL https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language Models are Unsupervised Multitask Learners. *OpenAI blog*, 1(8):9, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 21:140:1–140:67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. In-Context Retrieval-Augmented Language Models. *CoRR*, abs/2302.00083, 2023. URL https://doi.org/10.48550/arXiv.2302.00083.

Qiu Ran, Yankai Lin, Peng Li, Jie Zhou, and Zhiyuan Liu. NumNet: Machine Reading Comprehension with Numerical Reasoning. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing EMNLP-IJCNLP*, pages 2474–2484. Association for Computational Linguistics, 2019. URL https://doi.org/10.18653/v1/D19-1251.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 3980–3990. Association for Computational Linguistics, 2019. URL https://doi.org/10.18653/v1/D19-1410.

Stefan Riezler and John T. Maxwell III. On Some Pitfalls in Automatic Evaluation and Significance Testing for MT. In *Proceedings of the Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization@ACL 2005, Ann Arbor, Michigan, USA, June 29, 2005*, pages 57–64. Association for Computational Linguistics, 2005. URL https://aclanthology.org/W05-0908/.

Hajo Rijgersberg, Mark van Assem, and Jan L. Top. Ontology of Units of Measure and Related Concepts. *Semantic Web*, 4(1):3–13, 2013. URL https://doi.org/10.3233/SW-2012-0069.

Julian Risch, Timo Möller, Julian Gutsch, and Malte Pietsch. Semantic Answer Similarity for Evaluating Question Answering Models. *CoRR*, abs/2108.06130, 2021. URL https://arxiv.org/abs/2108.06130.

Stephen Robertson. On GMAP: and Other Transformations. In *Proceedings of the 2006 ACM CIKM International Conference on Information and Knowledge Management, Arlington, Virginia, USA, November 6-11, 2006*, pages 78–83. ACM, 2006. URL https://doi.org/10.1145/1183614.1183630.

Stephen E. Robertson and Karen Spärck Jones. Relevance Weighting of Search Terms. *Journal of the Association for Information Science and Technology*, 27(3):129–146, 1976. URL https://doi.org/10.1002/asi.4630270302.

Stephen E. Robertson and Hugo Zaragoza. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, 2009. URL https://doi.org/10.1561/1500000019.

Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. Okapi at TREC-3. In *Proceedings of The Third Text REtrieval Conference, TREC*, volume 500-225 of *NIST Special Publication*, pages 109–126. National Institute of Standards and Technology (NIST), 1994. URL http://trec.nist.gov/pubs/trec3/papers/city.ps.gz.

Thomas Roelleke, Jun Wang, and Stephen Robertson. Probabilistic Retrieval Models and Binary Independence Retrieval (BIR) Model. In *Encyclopedia of Database Systems*, pages 2156–2160. Springer US, 2009. URL https://doi.org/10.1007/978-0-387-39940-9_919.

Subhro Roy, Tim Vieira, and Dan Roth. Reasoning about Quantities in Natural Language. *Transactions of the Association for Computational Linguistics*, 3:1–13, 2015. URL https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/452.

Jon Saad-Falcon, Omar Khattab, Christopher Potts, and Matei Zaharia. ARES: An Automated Evaluation Framework for Retrieval-Augmented Generation Systems. *CoRR*, abs/2311.09476, 2023. URL https://doi.org/10.48550/arXiv.2311.09476.

Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic Routing Between Capsules. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 3856–3866, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/2cad8fa47bbef282badbb8de5374b894-Abstract.html.

Amrita Saha, Shafiq R. Joty, and Steven C. H. Hoi. Weakly Supervised Neuro-Symbolic Module Networks for Numerical Reasoning over Text. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 11238–11247. AAAI Press, 2022. URL https://ojs.aaai.org/index.php/AAAI/article/view/21374.

Swarnadeep Saha, Harinder Pal, and Mausam. Bootstrapping for Numerical Open IE. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Volume 2: Short Papers*, pages 317–323. Association for Computational Linguistics, 2017. URL https://doi.org/10.18653/v1/P17-2050.

Tetsuya Sakai. On Fuhr's guideline for IR evaluation. *SIGIR Forum*, 54(1):12:1–12:8, 2020. URL https://doi.org/10.1145/3451964.3451976.

Ruslan Salakhutdinov and Geoffrey E. Hinton. Semantic Hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009. URL https://doi.org/10.1016/j.ijar.2008.11.006.

Gerard Salton. A Vector Space Model for Information Retrieval. *Journal of the ASIS*, pages 613–620, 1975.

Gerard Salton and Chris Buckley. Term-Weighting Approaches in Automatic Text Retrieval. *Information Processing & Management*, 24(5):513–523, 1988. URL https://doi.org/10.1016/0306-4573(88)90021-0.

Gerard Salton and Michael E. Lesk. The SMART Automatic Document Retrieval Systems - an Illustration. *Communications of the ACM*, 8(6):391–398, 1965. URL https://doi.org/10.1145/364955.364990.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter. *CoRR*, abs/1910.01108, 2019. URL http://arxiv.org/abs/1910.01108.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. ColBERTv2: Effective and Efficient Retrieval via Lightweight Late Interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pages 3715–3734. Association for Computational Linguistics, 2022. URL https://doi.org/10.18653/v1/2022.naacl-main.272.

Sunita Sarawagi and Soumen Chakrabarti. Open-domain Quantity Queries on Web Tables: Annotation, Response, and Consensus Models. In Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani, editors, *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 711–720. ACM, 2014. URL https://doi.org/10.1145/2623330.2623749.

Sunita Sarawagi and William W. Cohen. Semi-Markov Conditional Random Fields for Information Extraction. In *Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada]*, pages 1185–1192, 2004. URL https://proceedings.neurips.cc/paper/2004/hash/eb06b9db06012a7a4179b8f3cb5384d3-Abstract.html.

Robin M. Schmidt. Recurrent Neural Networks (RNNs): A Gentle Introduction and Overview. *CoRR*, abs/1912.05911, 2019. URL http://arxiv.org/abs/1912.05911.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A Unified Embedding for Face Recognition and Clustering. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 815–823. IEEE Computer Society, 2015. URL https://doi.org/10.1109/CVPR.2015.7298682.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving Neural Machine Translation Models with Monolingual Data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016a. URL https://doi.org/10.18653/v1/p16-1009.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016b. URL https://doi.org/10.18653/v1/p16-1162.

Erhan Sezerer and Selma Tekir. A Survey On Neural Word Embeddings. *CoRR*, abs/2110.01804, 2021. URL https://arxiv.org/abs/2110.01804.

Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. REPLUG: Retrieval-Augmented Black-Box Language Models. *CoRR*, abs/2301.12652, 2023. URL https://doi.org/10.48550/arXiv.2301.12652.

Amit Singhal, Chris Buckley, and Mandar Mitra. Pivoted Document Length Normalization. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'96, August 18-22, 1996, Zurich, Switzerland (Special Issue of the SIGIR Forum)*, pages 21–29. ACM, 1996. URL https://doi.org/10.1145/243199.243206.

Xinying Song, Alex Salcianu, Yang Song, Dave Dopson, and Denny Zhou. Fast WordPiece Tokenization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 2089–2103. Association for Computational Linguistics, 2021. URL https://doi.org/10.18653/v1/2021.emnlp-main.160.

Hugo O. Sousa, Ricardo Campos, and Alípio Jorge. TEI2GO: A Multilingual Approach for Fast Temporal Expression Identification. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, CIKM 2023, Birmingham, United Kingdom, October 21-25, 2023*, pages 5401–5406. ACM, 2023a. URL https://doi.org/10.1145/3583780.3615130.

Hugo O. Sousa, Ricardo Campos, and Alípio Mário Jorge. tieval: An Evaluation Framework for Temporal Information Extraction Systems. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2023, Taipei, Taiwan, July 23-27, 2023*, pages 2871–2879. ACM, 2023b. URL https://doi.org/10.1145/3539618.3591892.

Georgios P. Spithourakis and Sebastian Riedel. Numeracy for Language Models: Evaluating and Improving their Ability to Predict Numbers. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 2104–2115. Association for Computational Linguistics, 2018. doi: 10.18653/v1/P18-1196. URL https://www.aclweb.org/anthology/P18-1196/.

Georgios P. Spithourakis, Steffen E. Petersen, and Sebastian Riedel. Clinical Text Prediction with Numerically Grounded Conditional Language Models. In *Proceedings of the Seventh International Workshop on Health Text Mining and Information Analysis, Louhi@EMNLP 2016, Austin, TX, USA, November 5, 2016*, pages 6–16. Association for Computational Linguistics, 2016. URL https://doi.org/10.18653/v1/W16-6102.

Daniel Spokoyny, Ivan Lee, Zhao Jin, and Taylor Berg-Kirkpatrick. Masked Measurement Prediction: Learning to Jointly Predict Quantities and Units from Textual Context. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 17–29. ACL, 2022. URL https://doi.org/10.18653/v1/2022.findings-naacl.2.

Daria Stepanova, Dragan Milchevski, Gerhard Weikum, Jannik Stroetgen, and Vinh Thinh Ho. *Device and Computer Implemented Method for Adding a Quantity Fact to a Knowledge Base*. Google Patents, 2023. US Patent App. 18/168,666.

Jannik Strötgen and Michael Gertz. HeidelTime: High Quality Rule-Based Extraction and Normalization of Temporal Expressions. In *Proceedings of the 5th International Workshop on Semantic Evaluation, SemEval@ACL 2010, Uppsala University, Uppsala, Sweden, July 15-16, 2010*, pages 321–324. The Association for Computer Linguistics, 2010. URL https://aclanthology.org/S10-1071/.

Jannik Strötgen and Michael Gertz. A Baseline Temporal Tagger for all Languages. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 541–547. The Association for Computational Linguistics, 2015. URL https://doi.org/10.18653/v1/d15-1063.

Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a Core of Semantic Knowledge. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 697–706. ACM, 2007. URL https://doi.org/10.1145/1242572.1242667.

Dhanasekar Sundararaman, Shijing Si, Vivek Subramanian, Guoyin Wang, Devamanyu Hazarika, and Lawrence Carin. Methods for Numeracy-Preserving Word Embeddings. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 4742–4753. Association for Computational Linguistics, 2020. URL https://doi.org/10.18653/v1/2020.emnlp-main.384.

Amir Vakili Tahami, Kamyar Ghajar, and Azadeh Shakery. Distilling Knowledge for Fast Retrieval-based Chat-bots. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pages 2081–2084. ACM, 2020. URL https://doi.org/10.1145/3397271.3401296.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, and Johan Schalkwyk et al. Gemini: A Family of Highly Capable Multimodal Models. *CoRR*, abs/2312.11805, 2023. URL https://doi.org/10.48550/arXiv.2312.11805.

Avijit Thawani, Jay Pujara, Filip Ilievski, and Pedro A. Szekely. Representing Numbers in NLP: a Survey and a Vision. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 644–656. Association for Computational Linguistics, 2021. URL https://doi.org/10.18653/v1/2021.naacl-main.53.

Ambler Thompson and Barry N Taylor. Use of the International System of Units (SI). *NIST Special Publication, Gaithersburg*, 2008.

OH Tittmann. The CGS System of Units. *Nature*, 45(1173):581–581, 1892.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving Retrieval with Chain-of-Thought Reasoning for Knowledge-Intensive Multi-Step Questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 10014–10037. Association for Computational Linguistics, 2023. URL https://doi.org/10.18653/v1/2023.acl-long.557.

Lewis Tunstall, Leandro Von Werra, and Thomas Wolf. *Natural language Processing with Transformers*. O'Reilly Media, Inc., 2022.

Naushad UzZaman, Hector Llorens, Leon Derczynski, James Allen, Marc Verhagen, and James Pustejovsky. SemEval-2013 Task 1: TempEval-3: Evaluating Time Expressions, Events, and Temporal Relations. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 1–9, Atlanta, Georgia, USA, jun 2013a. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/S13-2001.

Naushad UzZaman, Hector Llorens, Leon Derczynski, James F. Allen, Marc Verhagen, and James Pustejovsky. SemEval-2013 Task 1: TempEval-3: Evaluating Time Expressions, Events, and Temporal Relations. In *Proceedings of the 7th International Workshop on Semantic Evaluation, SemEval@NAACL-HLT 2013, Atlanta, Georgia, USA, June 14-15, 2013*, pages 1–9. The Association for Computer Linguistics, 2013b. URL https://aclanthology.org/S13-2001/.

Christophe Van Gysel and Maarten de Rijke. Pytrec_eval: An Extremely Fast Python Interface to trec_eval. In *SIGIR*. ACM, 2018.

C Van Rijsbergen. Information Retrieval: Theory and Practice. In *Proceedings of the joint IBM/University of Newcastle upon tyne seminar on data base systems*, volume 79, 1979.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017,*

December 4-9, 2017, Long Beach, CA, USA, pages 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Marc Verhagen, Robert Gaizauskas, Frank Schilder, Mark Hepple, Graham Katz, and James Pustejovsky. SemEval-2007 Task 15: TempEval Temporal Relation Identification. In *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*, pages 75–80, Prague, Czech Republic, jun 2007. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/S07-1014.

Marc Verhagen, Roser Saurí, Tommaso Caselli, and James Pustejovsky. SemEval-2010 Task 13: TempEval-2. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 57–62, Uppsala, Sweden, jul 2010. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/S10-1010.

Ellen M. Voorhees. The TREC Medical Records Track. In *ACM Conference on Bioinformatics, Computational Biology and Biomedical Informatics. ACM-BCB 2013*, page 239. ACM, 2013. URL https://doi.org/10.1145/2506583.2506624.

Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. Do NLP Models Know Numbers? Probing Numeracy in Embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP*, pages 5306–5314. ACL, 2019. URL https://doi.org/10.18653/v1/D19-1534.

Dingzirui Wang, Longxu Dou, Xuanliang Zhang, Qingfu Zhu, and Wanxiang Che. Enhancing Numerical Reasoning with the Guidance of Reliable Reasoning Processes. *CoRR*, abs/2402.10654, 2024. URL https://doi.org/10.48550/arXiv.2402.10654.

Yequan Wang, Aixin Sun, Jialong Han, Ying Liu, and Xiaoyan Zhu. Sentiment Analysis by Capsules. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 1165–1174. ACM, 2018. URL https://doi.org/10.1145/3178876.3186015.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Gerhard Weikum. Entities with Quantities. *IEEE Data Eng. Bull.*, 43(1):4–8, 2020. URL http://sites.computer.org/debull/A20mar/p4.pdf.

Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. End-to-End Neural Ad-hoc Ranking with Kernel Pooling. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, pages 55–64. ACM, 2017. URL https://doi.org/10.1145/3077136.3080809.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate Nearest Neighbor Negative Contrastive Learning for Dense

Text Retrieval. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=zeFrfgyZln.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 5754–5764, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/dc6a7e655d7e5840e66733e9ee67cc69-Abstract.html.

Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=B14TlG-RW.

Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. Optimizing Dense Retrieval Model Training with Hard Negatives. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, pages 1503–1512. ACM, 2021. URL https://doi.org/10.1145/3404835.3462880.

Xikun Zhang, Deepak Ramachandran, Ian Tenney, Yanai Elazar, and Dan Roth. Do Language Embeddings Capture Scales? In *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2020, Online, November 2020*, pages 292–299. Association for Computational Linguistics, 2020. URL https://doi.org/10.18653/v1/2020.blackboxnlp-1.27.

Wayne Xin Zhao, Jing Liu, Ruiyang Ren, and Ji-Rong Wen. Dense Text Retrieval based on Pretrained Language Models: A Survey. *CoRR*, abs/2211.14876, 2022. URL https://doi.org/10.48550/arXiv.2211.14876.

Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H. Chi, Quoc V. Le, and Denny Zhou. Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models. *CoRR*, abs/2310.06117, 2023. doi: 10.48550/ARXIV.2310.06117. URL https://doi.org/10.48550/arXiv.2310.06117.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/pdf?id=WZH7099tgfM.

Kun Zhou, Yeyun Gong, Xiao Liu, Wayne Xin Zhao, Yelong Shen, Anlei Dong, Jingwen Lu, Rangan Majumder, Ji-Rong Wen, and Nan Duan. SimANS: Simple Ambiguous Negatives Sampling for Dense Text Retrieval. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: EMNLP 2022 - Industry Track, Abu Dhabi, UAE, December 7 - 11, 2022*, pages 548–559. Association for Computational Linguistics, 2022. URL https://aclanthology.org/2022.emnlp-industry.56.

Justin Zobel, Alistair Moffat, and Laurence Anthony F. Park. Against Recall: Is It Persistence, Cardinality, Density, Coverage, or Totality? *SIGIR Forum*, 43(1):3–8, 2009. URL https://doi.org/10.1145/1670598.1670600.