Aus dem Deutschen Krebsforschungszentrum
(Vorstand: Prof. Dr. Michael Baumann, Ursula Weyrich)
Abteilung für Medizinische Bildverarbeitung
(Leiter: Prof. Dr. rer. nat. Klaus H. Maier-Hein)

Decentralized Infrastructure for Medical Image Analysis

The Development and Establishment of Kaapana as
an Open Framework for Imaging Platforms in Clinical Environments

Inauguraldissertation
zur Erlangung des Doctor scientiarum humanarum (Dr. sc. hum.)
an der
Medizinischen Fakultät Heidelberg
der
Ruprecht-Karls-Universität

vorgelegt von
Jonas Scherer
aus
Freiburg im Breisgau

2022

Dekan: Prof. Dr. med. Hans-Georg Kräusslich
Doktorvater: Prof. Dr. rer. nat. Klaus H. Maier-Hein

Contents

Acronyms

AE Application Entity

AI artificial intelligence

ANN artificial neural network

API application programming interface

BMBF German Federal Ministry of Education and Research

CaaS Containers-as-a-Service

CI Continuous Integration

CPU central processing unit

CTP RSNA MIRC Clinical Trials Processor

CT computed tomography

CV Computer Vision

DAG Directed Acyclic Graph

dcmqi DICOM for Quantitative Imaging

DCMTK DICOM Toolkit

DICOM Digital Imaging and Communications in Medicine

DICOM PM DICOM Parametric Map

DICOM SEG DICOM Segmentation Objects

DICOM SR DICOM Structured Reporting

DKFZ German Cancer Research Center

DKTK German Cancer Consortium

DL deep learning

DNS Domain Name System

eCRF electronic case report form

ES Elasticsearch

ETL Extract, Transform, Load

EU European Union

FaaS Functions-as-a-Service

FAIR Findable, Accessible, Interoperable and Reusable

FHIR Fast Healthcare Interoperability Resources

FL Federated Learning

GCP Google Cloud Platform

GDPR General Data Protection Regulation

GGO Ground-glass opacity

GPU Graphics Processing Unit

HIS Hospital Information System

HL7 Health Level 7

HTTPS Hypertext Transfer Protocol Secure

HTTP Hypertext Transfer Protocol

IaaS Infrastructure-as-a-Service

IAM Identity/Access Management

IBSI image biomarker standardisation initiative

IDC Imaging Data Commons

IDE integrated development environment

IHE Integrating the Healthcare Enterprise

IID Invoke Image Display

IOD Information Object Definitions

JIP Joint Imaging Platform

JSON JavaScript Object Notation

K8s Kubernetes

LDAP Lightweight Directory Access Protocol

LIMS Laboratory Information Management System

MGPD META General Purpose Dashboard

MIC Medical Image Computing

MITK Medical Imaging Interaction Toolkit

ML machine learning

MRI magnetic resonance imaging

NBAD Network Behavior Anomaly Detection

NCI National Cancer Institute

NEMA National Electrical Manufacturers Association

NIfTI Neuroimaging Informatics Technology Initiative

noSQL "Not only SQL"

Nrrd "nearly raw raster data"

NUM Network of University Medicine

OCI Open Container Initiative

OHIF Open Health Imaging Foundation

OIDC OpenID Connect

ONNX Open Neural Network Exchange

OS operating system

PaaS Platform-as-a-Service

PACS Picture Archiving and Communication System

PDF Portable Document Format

PET positron emission tomography

PHT Personal Health Train

PSMA Prostate-specific membrane antigen

QIICR Quantitative Image Informatics for Cancer Research

RACOON Radiological Cooperative Network

RAM random-access memory

RBAC role-based access control

REST Representational State Transfer

RIS Radiological Information System

RöKo Röntgenkongress

RSNA Radiological Society of North America

RTE runtime environment

SaaS Software-as-a-Service

SATORI "Segmentation and Annotation Tool for Radiomics and Deep Learning"

SEG Segmentation Objects

SPARQL SPARQL Protocol And RDF Query Language

SPA single-page application

SOA service-oriented architecture

SSO single sign-on

SVM support vector machine

TCIA the Cancer Imaging Archive

TFDA Trustworthy Federated Data Analytics

UID unique identifier

UI user interface

URL Uniform Resource Locator

US ultrasound

UTC Coordinated Universal Time

VM virtual machine

VNC Virtual Network Computing

VR Value Representation

XML Extensible Markup Language

List of Figures

List of Tables

1 | Introduction

More than 125 years ago, on November 8, 1895, Wilhelm Conrad Röntgen discovered the X-rays, which shortly afterwards were used to X-ray a human being for the first time. This famous image represents the first radiography of a human being, which shows the hand of Röntgen's wife, Bertha, with all its bones clearly visible. This fact marks the beginning of modern medical imaging and radiology (see figure 1.1). The field has been developing rapidly ever since; what back then required long exposure times of several minutes can now be acquired in milliseconds. What used to be simple two dimensional X-rays, has been extended by a variety of other imaging modalities such as computed tomography (CT), magnetic resonance imaging (MRI), positron emission tomography (PET) or ultrasound (US) and revolutionized modern medical care over the years. Today, medical imaging plays a central role in medical care and is used in a wide range of applications - from the detection and diagnosis of diseases, over the classification and staging of progression, to the treatment of a particular condition. Thanks to the technical nature of radiology, another development was able to emerge here very early on: **The digital revolution.**

Figure 1.1: *First medical X-ray of the hand of Röntgen's wife Anna Bertha Ludwig.*

1.1 The Digital Revolution and Radiology

What still is a dream of the future in many areas of medical care has long been common practice in radiology and nuclear medicine. Almost all imaging is already digitally acquired and managed today. Moreover, the internationally widely used standard for Digital Imaging and Communications in Medicine (DICOM) was published by the National Electrical Manufacturers Association (NEMA) for the first time as early as 1985, which shows how early this field has dealt with the topics of digitization and standardization [NEMA, 2021d]. This early and deeply integrated digitization, along with the application of imaging in daily care, has led to an explosion in the amount of data, which will become even more pronounced in the future [Dash et al., 2019]. Due to the variety of modalities and protocols available in today's scanners, a vast amount of data can be collected for every single patient and in case of MRI or US this can even be done without ionizing radiation, and thus presumably without involving any health risks. This development should accommodate today's Information Age enabling new data-driven systems to take radiology to the next level. But does all this data really already provide a lot of new information?

What at first seems like a major progress of digitization, still involves great challenges on closer inspection. The huge amounts of existing data do not necessarily offer additional value - the information it contains must first be extracted from the data so that its potential can unfold. Although digital imaging has certainly enabled many useful new tools and better image quality, the core of the interpretation still remains on the human specialist, who based on expertise and experience derive the conclusions from the images. This results in two hurdles for the evaluation of large volumes of data.

On the one hand, manual inspection limits the speed and scale at which data analysis can be performed. Experts are rare, expensive and moreover, the practical tasks involved with manual screening are often monotonous and repetitive, so the risk of human error increases [Bercovich and Javitt, 2018]. Furthermore, the findings are often subjective due to differences in experience or skill and may well differ from expert to expert. The limit for human interpretation of the data has been reached, resulting in useful image data being generated but never actually utilized [Bercovich and Javitt, 2018]. On the other hand, a big challenge associated with manual interpretation has been well summarized by Gillies et al.: "Images Are More than Pictures, They Are Data" [Gillies et al., 2015]. In other words, besides the visual information accessible to humans, there might be additional information hidden in images. Because these connections are often very complex and subtle, human observers may simply not be able to identify them. These hidden image features could lead to the discovery of even deeper insights, possibly far beyond the current capabilities of manual interpretation.

The solution to these hurdles seems obvious - computers could simply be used to overcome them. Indeed, computers and algorithms have been utilized in medical imaging for many decades, and most modern modalities would not be possible without digital post-processing of acquired data or the computer-aided control of such complex machines. While this has been very successful for the development of new imaging technologies, analysis tools and the improvement of image quality, the situation is much more difficult when it comes to the digital interpretation of images. Until recently, images were analyzed using what are now called traditional image processing methods. Here, mainly pixel-wise operations are performed which, for example, increase the contrast, emphasize certain edges or reduce noise. Based on the resulting adjusted data, which highlights only the important parts, rule-based systems were used to try to derive new information from the images. Typical tasks of such systems are, for example, object detection, segmentation, classification or image registration. Due to the high complexity and variability of most medical image data, the capabilities of these methods have always remained limited, falling far short of human experts. Generally, not only in the medical context, it has proven very difficult to "teach computers to see" and thus to understand the content of images. This has led to the rise of Computer Vision (CV), an own field of research that addresses these challenges.

1.2 Medical Image Computing and Machine Learning

For some time now, however, another technique has been in the focus of CV, which seems to be better suited for the analysis of images than the procedure with the hand-crafted adjustments and rules - which is **machine learning (ML)**.

Put simply, the difference here is that instead of specifying concrete processing steps and decision rules, computer models learn from sample data. This applies in particular to **deep learning (DL)**, a ML method that has revolutionized the field since around 2012. Since then, not only many of the organized international scientific competitions could be won by DL-based approaches, but also the best results of previous years have been significantly outperformed.

Most of the prediction models are trained on the basis of training data, which already includes the solution for the respective problem by annotations (ground truth). In this process, all available training samples are iteratively cycled through to produce predictions of the computer model. If this prediction is wrong, the model gets automatically adjusted in such a way that the probability of the next prediction with the same sample is higher to be correct. This process of repeatedly going through the data and optimizing the model is called "training" and, optimally, leads to better and better predictions as the training proceeds.

DL uses models created by concatenating millions of artificial neurons arranged in layers to form artificial neural networks (ANNs) [Yegnanarayana, 2009]. Such models have so many parameters that even very complex problems that previous methods were unable to cope with, can be tackled successfully. Even though some of the basic techniques of DL had already been known for decades, it was mainly the transfer of the computations to new hardware in the form of graphics cards and their Graphics Processing Units (GPUs) that enabled a dramatic acceleration of the training process and thus initiated the current boom in artificial intelligence (AI).

It didn’t take long for the new techniques to be applied to medical imaging, which also led to vast advances in image analysis. Especially, tasks like segmentation, object recognition and classification could be automated much more accurately. Geoffrey Hinton, a well-known scientists in the field of artificial intelligence, even went so far as making the following provocative statement in 2017:

„It’s just completely obvious that in five years deep learning is going to do better than radiologists, it might be ten years.“ [Mukherjee, 2017]

During a lecture at a hospital, he even recommended: "They should stop training radiologists now." [Mukherjee, 2017] But are algorithms now really better at interpreting images than radiologists?

Certainly not. However, this technology truly offers great potential to extract hidden information from images and to make it available to radiologists in a supportive way, allowing them to better cope with the large amounts of data and using the additional information to make better medical decisions. For example, it might be possible to assess straight from the image whether an abnormal finding is a benign or malignant alteration - which would save patients from psychological and physical health risks as well as health care systems from costs by avoiding unnecessary biopsies [Jäger et al., 2017]. This is just one application example among many where new automated analysis techniques could help improve medical care.

1.3 Opportunities

In general, the areas of application with the greatest potential can be grouped as follows [Kleesiek et al., 2020]:

1.3.1 Detection and Diagnosis

ML disciplines, such as classification, detection, segmentation or registration could be used to automatically identify suspicious areas in images and bring them to the attention of the specialist. Additionally, they could be used to classify cases and thus help in making a diagnosis. The larger the data sets become, the more difficult it will be to sift through them manually. An automatic pre-analysis for anomalies - or a double-check of the manually identified areas - could become very important here.

1.3.2 Characterization and Personal Medicine

In the future, personalized medicine will become increasingly important for the successful treatment of highly individual disease profiles. This involves efforts to find tailored treatment strategies for very small groups of patients through increasingly fine-tuned characterization of the medical conditions. For this purpose, along with genetic markers, imaging-generated biomarkers could also help to enable this fine subdivision of patient cohorts to facilitate highly individualized therapies.

1.3.3 Monitoring

For many indications, especially for cancer treatment, it is important to monitor the development of the disease on a regular basis. Medical image analysis can support this process, for example, by providing more comprehensive and better surveying tools to evaluate the progression or the effectiveness of a treatment [Kickingereder et al., 2019]. Consequently, for example, a non-response to therapy could be identified earlier, and thus adjusted more quickly.

1.3.4 Prognosis and Prediction

Imaging could also contribute to the prediction of appropriate and promising therapies. Models could be trained to learn from the experience of many previous treatments to allow an assessment of success for a particular patient even before they are applied. Predictions could also be made for the likely progression of the disease, like tumor growth, to provide a better basis for decision-making by physicians [Petersen et al., 2019].

1.3.5 Knowledge Transfer, Costs and Standardization

Beyond the benefits that could translate directly to better medical treatment, there are also opportunities in other sectors. Faster, better, and more preventive medical care could for example also lead to cost savings for health systems. By providing objective information based on algorithms, the current state of knowledge and technology could be transferred much more easily, quickly and cost-effectively to areas of the world where access to it is currently not so easy. And last but not least, this could lead to a higher standardization for medicine, and especially the technical management of it, which again would facilitate the development of new methods and a higher level of care in general.

1.4 Open Challenges

Now that the versatile potential of this promising technology has been presented, the question is if and how this can actually be leveraged. What are currently the biggest obstacles and hurdles affecting the research and development of these methods?

First, the advancement of the methods themselves - for example, the architecture of the networks, new layers, or other training approaches - is required to achieve the necessary safety and interpretability of the predictions for their application in medical care. The lack of understanding of how these methods finally come to their prediction is a major problem for medicine, since physicians are ultimately responsible for their patients and therefore may not simply rely on an automatic prediction, which they cannot comprehend and from where they cannot rely on the confidence interval. DL models are often black boxes that deliver reliable results, as statistical analyses show, but the way in which these decisions are made is extremely hard to track (Black Box Problem).

Second, it has been shown that existing methods can already provide very good results when adequate data is available. As described earlier, these models require a huge number of parameters to handle such complex issues. Although this gives the necessary space to be able to map and learn even very sophisticated relationships, it also means that there is a lot of space to just learn the training cases "by heart". This problem, which is called overfitting, means that an algorithm does not learn to solve the given task, but simply ends up recalling the given training examples. If this model is now confronted with new cases, it will not be able to process them correctly. To address this problem, it often takes large amounts of sample data to prevent such memorization and overfitting effects. Furthermore, the number of available examples representing the different characteristics of the task to be learned should be about the same in order to avoid a class imbalance. As a result, even for the rather rare variants, a considerable number of examples are needed, which are often difficult to find. At the latest, when it comes to personalized medicine with its tiny sample sizes, the challenges cumulate with the required data volumes.

So far, the methods and associated models are currently developed and trained mostly by data scientists working at universities, institutes or private companies. For this purpose, if the data is not generated and therefore available at the own institution, collaborations are formed with partners which have at least small volumes of data in question available. Then, everything is usually pooled from the originating sites in order to train a model centrally, which means that data owners have to give the data out of their control, what in turn results in both legal and incentive hurdles.

Many ML models are still trained "supervised", which means that the ground truth must be provided. Since the creation of medical annotations can often only be done by experts, the process is quite expensive and the data is therefore valuable. Moreover, medical data - especially images - are highly sensitive personal records, which are subject to strict data protection requirements for good reasons. The sharing of data can therefore also cause legal challenges, as especially within the European Union (EU) the General Data Protection Regulation (GDPR) imposes particularly strict guidelines in this regard. Anonymization cannot be easily achieved with images either, since many of the image features remain highly individual. While in the past the removal of personal metadata from images appeared to be an effective way for anonymization, later techniques have shown that a person can easily be identifiable by the raw image information alone. The learning from this is that what is considered sufficiently anonymous today may not be so tomorrow.[Floca, 2014, Rocher et al., 2019, Ravindra and Grama, 2021]

Training and evaluation of new methods has so far typically been based on scientific competitions, which are normally organized by international conferences and which provide the corresponding datasets. The retention of a subset of the total dataset, which is not made available to the participants, allows the evaluation of submitted methods and their associated trained models at the end of the competition. Hold-out testing data ensures that the evaluation is based on previously unpublished data, which should provide a more realistic picture of the method’s performance. However, this approach also has disadvantages in terms of the significance of the results.

While such competitions provide a good comparability and overview for evaluating the alleged state of the art, they do not necessarily reveal a lot about how well these methods perform in real clinical practice. This is due to small, homogeneous datasets that often originate from a single source only [Maier-Hein et al., 2018], which does not reflect the heterogeneity of data in clinical practice. The methods can then be extremely fine-tuned to work well with the given data, which in turn results in very good evaluation results, being this typical overfitting problem. At first glance, the excellent scores achieved at these contests suggest that many of the existing problems have already been solved - in practice, however, the picture is quite different. This problem has already been addressed by providing more diverse datasets for more realistic simulations - at least, the test data used for evaluation should have different characteristics than the training data in order to allow a better assessment. This is not possible for many of the given tasks, though, because of the unavailability of matching public data.

1.5 Approach

One of the greatest problems in developing predictive and robust models in medical imaging is the lack of high-quality data and, in particular, its collection in a central location. This is important because this data does exist - it's merely not accessible.

1.5.1 How to overcome these challenges?

Many hospitals, and particularly university hospitals, have accumulated a lot of high-quality data through their daily work with imaging. Consequently, reducing or eliminating the technical and legal barriers for collaborations between data scientists and clinicians to make use of this data for research could be a solution to such challenges. The main research question addressed in this thesis is therefore:

Is it possible to shift both the evaluation and the training of these methods to the clinics, instead of fetching the data?

In other words, can the algorithms be shared with hospitals instead of collecting the data at research institutes? By ensuring that data owners never have to disclose their information and thus always retain control over it, data protection and privacy requirements are significantly lower. Furthermore, the importance and impact of this research could be increased through much larger, multi-center studies, which will be an incentive for the researchers involved. The vision of such a decentralized data science for medical imaging holds numerous advantages. Besides a realistic evaluation of research results in everyday clinical life, this could also accelerate the translation from bench to bedside, as the technology is already established in the clinical environment. The required strong collaboration of data and medical experts could also improve both sides' understanding of the issues and thus lead to more interdisciplinary knowledge.

What is needed to make this feasible?
The highly sophisticated and specialized technical nature of both medical imaging and the data-driven analysis of it makes an on-site decentralized technical infrastructure necessary. This infrastructure should provide the hospitals with the necessary computing hardware, interfaces to the local data systems and also an execution environment for algorithms. Also, more standards for data processing, such as defined processing pipelines, need to be developed.

Since this is a rather novel topic and there was not much preliminary work to build on at the time of launch, a step-by-step and iterative approach was taken in order to be able to assess experience and actual requirements of such a system. This was achieved by leveraging the existing network of several German university hospitals, which have joined forces within the German Cancer Consortium (DKTK) [DKTK, 2021]. First, the departments of radiology and nuclear medicine of the DKTK partner sites have linked together through the Joint Imaging Platform (JIP) project, which aims to strengthen the collaboration between the various sites and to implement and establish the idea of such a decentralized and distributed infrastructure for medical imaging.

1.5.2 Objectives

The main goal of this work was the realization of a distributed digital infrastructure for medical imaging, which allows the evaluation of the latest research findings in real clinical environments. Here, the high-quality data available at hospitals should become accessible for scientific research without endangering the privacy of the individual-related data by always leaving the data in the hands of the owners. Multi-center radiological studies have a special focus here, as the collaboration of several centers is challenging but can also have great advantages for research. The resulting large and heterogeneous datasets should enable more realistic evaluations with data from real-world clinical practice.

The technical implementation of this infrastructure is supposed to facilitate this process by establishing uniform hardware setups, interfaces and processing procedures. For this purpose, an infrastructure software should be developed, which is open-source an such available to the whole community. The basic pillars of the targeted system, such as servers, technology stack and applications, should provide a solution for the operation of local platform instances and allow new research outcomes to be applied, evaluated and enhanced directly on-site of hospitals and with access to their local datasets. So besides the technical implementation, the goal was also to establish and maintain operational instances of this infrastructure within the German university clinics and to support concrete research projects.

Since science and the domain in general is evolving rapidly and technical demands are constantly changing, the goal was not to create a dedicated project specific infrastructure, but rather to provide a flexible, extensible, and customizable framework. Characteristics such as modularity, extensibility, and reusability should ensure that the software can be used for many future research projects and use cases, paving the way for collaboration and standardization within the community and also providing

a realistic basis for federated approaches. In general, it should provide everything needed to build project-specific software platforms while ensuring that the different platforms remain compatible with each other. This software framework is called **Kaapana** and will be referenced as such in the following.

1.5.3 Requirements

The requirements for such an infrastructure were elaborated in collaboration with the medical experts of the JIP project. For this purpose, all 21 radiology and nuclear medicine departments of the clinical DKTK partner sites were involved and asked [Scherer et al., 2020b]. This resulted in a rather heterogeneous landscape in terms of patient numbers, primary modalities and local IT systems in the different hospitals. Two main use cases for such a system emerged from the partner consultations. First, the need to enable and support multicenter imaging studies with access to larger numbers of cases for retrospective data analysis emerged and to facilitate collaboration in the field of medical imaging in general. Secondly, the focus was on improved integration of the existing data processing, annotation and sharing tools into the clinical environment. In particular, machine learning technologies and federated data analytics were of interest. These technical, organizational, and legal demands of our medical partners, along with the expectations and requests of the data-science community, were translated into the following requirements:

Integrability: The fundamental principle of local execution of analytical methods as an enhancement of the existing clinical infrastructure is central. Consequently, the platform should be integrated tightly with the existing local clinical systems and the user interaction should be as close as possible to the established clinical workflows and tools. Therefore, communication with the clinical IT systems, especially the Picture Archiving and Communication System (PACS), is very important. It should be possible to transfer images directly from the hospital's imaging archive into the platform. Running software within protected clinical IT environments requires the servers to be isolated and no external connections should be necessary during daily operation.

Data accessibility: Data analysis should never interfere with the original image data of the clinic and should only be executed on redundantly stored copies on the server located at the respective site. In order to achieve high compatibility between different hospitals and their systems, the widely established standard DICOM should be utilized whenever possible. This means that besides the images themselves, also processing results like segmentations or Radiomics features should be converted and stored as DICOM files. Since dealing with sensitive data requires strict data access control,

there should be a user, role and authentication service, which integrates well with locally established identity providers.

Algorithmic accessibility: The infrastructure should foster and facilitate the spread and application of cutting-edge methods from research with clinical partners and within the community. This requires a versatile and efficient integration path for in-house developments to make them compatible with the framework, supporting arbitrary processing steps, using different programming languages and input/output formats etc. Once integrated, these algorithms or extensions should also be easily shareable between users and instances of Kaapana compatible platforms.

Data sovereignty: Although the infrastructure is intended to be used for the collective analysis of data from different participating sites, data sovereignty must remain exclusively with the data owners. The design does not foresee the transmission of any imaging data for now, but should allow this if it is explicitly desired and needed for a project. However, for some use cases, for example Federated Learning (FL), it may be useful and necessary to share certain bits of information. This is less about the sharing of imaging data, but rather aggregated analysis results or trained machine learning models. Therefore, the framework should also provide solutions to enable secure and controlled exchange of those.

Data exploration and cohort specification: Since analysis methods are usually designed for very specific types of input data, exploration and filtering of an existing archive is very important. The available data should be presented in a visually structured way, to quickly get an overview of image properties such as modality, protocol, examined body parts or patient characteristics. This information should then also be used to filter the available data according to specific search criteria in order to pass the resulting cohort to an analysis pipeline for processing. In addition to the metadata, a way to visually view the medical images and their annotations should also be provided.

Maintenance and future-proofness: As decentralized systems require multiple individually operated instances to be maintained, the installation and maintenance must be possible by non specialized technical staff of each site. Where possible, already established and standardized processes for server setup and maintenance from other areas should be utilized. For high reusability, a modular design is desirable. Here, high scalability should also be considered in order to be able to process large amounts of data in parallel. Since a future linking of the different instances could be desired, this should also be technologically possible.

1.6 Outline

The thesis begins with a review of related work in the field and its approaches, capabilities, and limitations. As decentralized infrastructure for medical imaging is a relatively new field of research, the preliminary academic groundwork is limited. For this reason, publications from the separate requirement domains of Integrability, Data Accessibility, Algorithmic Accessibility, Data Sovereignty, and Data Exploration/Specification are reviewed individually first. Afterwards, existing solutions for comparable imaging platforms are also examined and outlined. The following methods chapter first explains the main concept of the infrastructure, its components and the integration with the hospital IT landscape. It also discusses the stakeholders and exemplary use cases that highlight the necessary capabilities of the framework. The realization of this is then described through the explanation of the technology stack and the implementation and workings of the components themselves. By introducing the core building blocks, the various functional sections of the platform are presented. At the end of this chapter, details on server setup and installation as well as the designed build system and continuous integration are given. The following chapter of the experiments and results, first addresses the evaluation of requirements. The DKTK and RACOON consortia and their respective infrastructure configurations are also described at this point, since they serve as evaluation environments for the developed software. Then, the use cases are examined for their feasibility on the basis of concrete applications and implementations. In the discussion, the results are reviewed critically and deficiencies or suggestions for improvement are made. Here, particular attention is paid to the realization of data accessibility and exploration as well as algorithmic accessibility, server management and security. Finally, the approach and what has been achieved are briefly summarized and an outlook is given on possible future developments.

2 Related Work

There are already numerous software projects designed to enable and facilitate research on and with medical images. In fact, many of the sub-requirements of this work already have a wide range of established prior work that has been considered for the development of Kaapana. The challenge here was to adapt already established and successful technologies into the infrastructure in such a way that they work and communicate natively with each other, and also to combine tools from different domains such as imaging and the field of deep learning. Thus, for the required tasks, the wheel should not be reinvented and everything re-implemented, but rather existing tools from the community should be reused as much as possible. Since this is currently a relatively new but also very rapidly advancing field of research, it is very difficult to give a conclusive overview of all projects and their development status at a given time. For this review, the various requirement categories are first addressed individually with related work, followed by an overview of existing infrastructure/platform efforts that have a similar mission.

2.1 Integrability

The challenge with integrating data processing into hospitals stems mainly from the fact that the two communities of everyday clinical practice and data scientists operate separately from each other.

In the clinical context, DICOM is primarily used for the management and transmission of imaging data. It is an open standard which has been developed since the 1980s and, besides the definition for the raw image information, also defines metadata, e.g. regarding the data acquisition or patient. The great advantage here is that this standard is internationally widely used and supported by practically every medical imaging manufacturer. As such, this is the format produced by the scanners and is also used for storage in the clinical PACS. Consequently, for successful integration into hospitals, this standard should be supported so that images can be transferred directly from clinical systems to the processing infrastructure.

In this context, the RSNA MIRC Clinical Trials Processor (CTP) [Erickson et al., 2014, RSNA MIRC, 2021] was developed as a software project that facilitates typical requirements for the handling of imaging data in clinical studies. By opening a DICOM receiver and enabling simple processing pipelines, the CTP allows images to be received from clinical systems and to be further processed by forwarding, anonymization or other processing steps.

The DICOM Toolkit (DCMTK) [Eichelberg et al., 2004, OFFIS, 2021a] also offers a comprehensive collection of tools, which allow, for example, the sending of DICOM files, the extraction of metadata or the generation of new DICOM objects. Strict adherence to the standard ensures that the resulting files are also compatible with other systems, such as the clinical PACS.

2.2 Data Accessibility

For redundant data storage of medical images, research PACS are usually used, which are operated in parallel to the clinical production systems. For this purpose, open and free software projects such as Orthanc [Jodogne, 2018], ConQuest [Conquest, 2021] or Dcm4che [Gunter Zeilinger, 2021] are often used, which have already been developed and used for many years. While these projects mainly focus on the storage and management of DICOM-based data, there are also projects such as XNAT [Herrick et al., 2016] which, apart from supporting arbitrary file formats, also have some tools for the automatic processing of data. These projects from the community offer full support of the DICOM interfaces and enable storing and querying of the datasets.

Although DICOM offers many advantages for clinical use, vendor-specific customizations and enhancements over the years made handling this data more difficult for data scientists. This, among other factors, has led to the prevalence of other formats in medical imaging data science, such as the Neuroimaging Informatics Technology Initiative (NIfTI) [Initiative, 2011] and the "nearly raw raster data" (Nrrd) [teem, 2021] file formats. The ability to convert clinical data from DICOM to these formats and vice versa is therefore essential for data processing. For the conversion of DICOM, software toolkits and libraries such as the Medical Imaging Interaction Toolkit (MITK) [Wolf et al., 2005], Pydicom [Darcy Mason, 2021] or NiBabel [nipy.org, 2021] offer versatile possibilities to transform the data into desired formats.

For more exotic, less image-focused use cases, such as extraction of DICOM metadata, wrapped Portable Document Format (PDF) or Extensible Markup Language (XML) files, DCMTK offers many solutions. Existing projects are also available for the opposite conversion of processing results. DICOM for Quantitative Imaging (dcmqi) [Fedorov et al., 2016, Quantitative Image Informatics for Cancer Research, 2021] provides various libraries to facilitate the generation of standard-compliant DICOM Segmentation Objects (DICOM SEG) [NEMA, 2021a] from NIfTI masks and also the creation of DICOM Structured Reporting (DICOM SR) [NEMA, 2021e].

2.3 Algorithmic Accessibility

Standardization is also helpful for the applicability and shareability of new methods. Existing implementations of analysis methods for medical image processing come in a wide variety of programming languages and frameworks. This often requires very specific execution environments that meet the necessary dependencies or even operating system requirements, which often makes sharing and distribution difficult. Differing implementations also lead to less comparable results, as for example is the case with radiomics-based imaging biomarkers. If the corresponding features were not implemented uniformly across different projects, the characteristics of the extracted features may also differ. For this case, the image biomarker standardisation initiative (IBSI) created standards that should avoid these problems and allow uniform results [Zwanenburg et al., 2020]. These standards have already been adopted, in projects such as MITK Phenotyping [Götz et al., 2019] or PyRadiomics [van Griethuysen et al., 2017], which should produce the same results with different implementations and thus increase accessibility.

Similarly, in the context of DL, certain frameworks have emerged to facilitate building and working with the complex models. Tensorflow [Developers, 2021], Pytorch (lightning) [Falcon et al., 2020, Paszke et al., 2019] or Keras [Chollet et al., 2015] are prominent examples that unify core tasks like model architecture or the training process for DL. Although the frameworks take different approaches for the implementation of DL networks, efforts such as the Open Neural Network Exchange (ONNX) [Bai et al., 2019] attempt to establish interoperability that allows models to be exchanged between frameworks. This allows, for example, the sharing of trained models of the latest research findings, and thus also increases the accessibility of them. However, this interoperability is not yet at a level where support for a single framework is sufficient, and the fact that the research community uses different ones implies that support for most of them should be feasible.

2.4 Data Sovereignty

The idea of sharing the analysis methods and algorithms rather than pooling the data has been around for several years. Especially in the medical context, where ethical-legal aspects of the data are of great importance, this approach was considered early on.

With DataSHIELD [Gaye et al., 2014], an infrastructure was presented in 2014 that was dedicated to this topic and realized the decentralized statistical analysis of health data. In this case, the analysis was limited to a statistical analysis of 96 harmonized variables conducted at several sites. With euroCAT [Deist et al., 2017], an infrastructure was presented that not only distributes a statistical analysis, but also trains models of a support vector machine (SVM) for the prediction of post-radiotherapy dyspnea at the sites. These models were evaluated using a five-fold cross-validation by using data from four sites for training and data from a fifth for testing. The integration of clinical data into the infrastructure is problematic here, since the approaches presented so far have involved manually preparing the input data at each site and making it available for analyses via prepared spreadsheets, for example.

This has been addressed by KETOS [Gruendner et al., 2019], another infrastructure for decentralized analysis of statistical data, by integrating the Health Level 7 (HL7) Fast Healthcare Interoperability Resources (FHIR) [HL7, 2019] to provide a standardized interface to query patient data. Furthermore, additional options for statistical analysis were added, such as Python and Jupyter Notebooks [Kluyver et al., 2016], in addition to DataSHIELD. The underlying technology used here includes Docker containers [Merkel, 2014], which enable reproducible data analysis and development environments. This concept has been enhanced in the Personal Health Train (PHT) [van Soest et al., 2018] by transferring such containers as "trains" from site to site to perform an analysis or to train a model, which also enables more sophisticated distributed learning approaches, where a model is continuously trained by passing it on.

The principle of Findable, Accessible, Interoperable and Reusable (FAIR) [Wilkinson et al., 2016] data is central to this process by preparing the input data accordingly. The PHT has already been used to perform several analyses [Deist et al., 2020, Deist et al., 2017, van Soest et al., 2018] in a distributed manner - this also includes an imaging-focused publication using Radiomics features [Shi et al., 2019]. Important here is that all of these projects and infrastructure deployments used textual data only for statistical analysis or the training of ML models. The use of raw data, as is necessary for medical image processing, is not the focus of any of these approaches so far.

2.5 Data Exploration and Cohort Specification

The concepts presented so far have mostly avoided the challenges of data exploration and selection by involving manual preparation of the experiment and associated curation and preprocessing of the data. This is suboptimal since it is time-consuming and requires a good understanding of the available data in order to plan experiments and develop new ideas. However, support for FHIR in KETOS enables standardized retrieval of data from clinical systems and the subsequent translation of received data into the OMOP Common Data Model enables [OHDSI, 2021] consistent access to information for analytical methods.

Since this thesis addresses imaging, where the DICOM Information Model [NEMA, 2021b] provides a standard with rich meta-information, it should be utilized for exploration and data selection. For the viewing of images, segmentations and reports, the Open Health Imaging Foundation (OHIF) [Ziegler et al., 2020] has published an open web-based viewer [Urban et al., 2017], which is already widely used in the community. By directly interfacing with a (research) PACS via DICOMweb [Genereaux et al., 2018], data can be visualized easily and quickly within a browser. However, data exploration for ML requires not only image visualization but also an overview of the available datasets providing information about image properties such as modality, examined body regions, or series descriptions. PACS typically use relational databases for metadata management and are usually limited to a small subset of the available information [Rascovsky et al., 2012]. The established open PACS from the community such as Dcm4chee or Orthanc have very limited user interfaces and offer few options to export the specified data selection.

To improve the searchability of the data, approaches have been developed to store the entire metadata in a "Not only SQL" (noSQL) database [Rascovsky et al., 2012], or even to convert it into an ontology, which can then be searched using the SPARQL Protocol And RDF Query Language (SPARQL) [Brunnbauer, 2013, VAN SOEST et al., 2014], for example. Where these solutions provide a better way to manage large amounts of images by using query languages to specify the desired metadata of a cohort, they still lack a visual way of data exploration. It must be known in advance what kind of data is available and how it can be characterized. The National Cancer Institute (NCI) Imaging Data Commons (IDC) [Fedorov et al., 2021], on the other hand, developed a portal based on Apache Solr [Apache, 2021c] and Google BigQuery [Google, 2021] that also enables visual and interactive cohort selection. This portal is well suited for data access of large online image databases such as the Cancer Imaging Archive (TCIA), but cannot be used for local on-premise setups due to its dependency on the Google Cloud Platform (GCP) [Google, 2021]. The preliminary work that was

ultimately used as the basis for Kaapana was presented by Goch et al. [Goch et al., 2018] and uses Elasticsearch (ES) as a database in combination with Kibana [Elastic, 2021] as a visualization tool to visualize the data sets. However, the export of the data defined by the queries has not been addressed here.

2.6 Existing Imaging Platforms

Projects that aim to combine these features to realize an infrastructure like the one envisioned in this thesis are rather rare. At the forefront are commercial products from major vendors such as Philips, Siemens and Nvidia. With Teamplay , Siemens Healthineers has a hybrid solution that uses a teamplay receiver within the hospital IT to transfer the relevant images to the "teamplay digital health platform", which is centrally hosted in a cloud [Healthineers, 2020]. In this central instance, proprietary so-called AI-Rad Companion Services then enable modern ML-based analysis methods on the images, followed by transferring the results back to the clinic. In contrast, Philips IntelliSpace Discovery [Philips, 2019] can be operated locally on-premise and, in addition to data visualization, also has a development environment, a runtime environment and a data management system, and thus appears to be very similar in structure to the infrastructure presented in this thesis. Furthermore, there is the possibility to share methods and algorithms with the community via the IntelliSpace Discovery Store. Since these are commercial products, the information is based only on manufacturers' brochures and is therefore difficult to verify and compare.

Clara Imaging [NVIDIA, 2019, NVIDIA, 2018] is also a closed-source clinical imaging framework from Nvidia, but so far it is free and has not been commercialized. Through the cooperation of several open-source projects from the community, such as the MITK Nvidia Annotation Plugin [MITK, 2019], the Slicer AI-assisted segmentation extension [Lasso, 2019] or OHIF [NVIDIA, 2020], the framework is research-oriented, where it is also active in research consortia such as Monai [MONAI initiative, 2021]. With "AI-Assisted Image Labeling", "AI Model Training" and "AI inference", NVIDIA Clara Imaging provides annotation, training and inference solutions for medical image data. Using a receiver, it is also possible to transmit data from the clinical systems for the training or prediction. Overall, this framework is less integrated, since, for example, MITK or Slicer are used as desktop software for annotation, and there is also no universal user interface yet.

QuantMed [Klein et al., 2020], developed by Fraunhofer MEVIS, is also described as an infrastructure for medical image processing with focus on data annotation and training / inference of ML models. The integrated web-based "Segmentation and Annotation Tool for Radiomics and Deep Learning" (SATORI) also allows the export of annotations in DICOM SEG. Due to the requirement of an open infrastructure developed and maintained by the community, these existing platforms have been considered in terms of concepts, base technologies and architectures (as far as they have been published) for the design of Kaapana, but not further evaluated due to the non-open source-codes and licenses.

RadPlanBio [Skripcak et al., 2016], on the other hand, is an open-source and free platform for multicenter image-related study management from the field of radiation therapy, which, like Kaapana, was developed mainly within the context of DKTK. The focus here is on the collection and management of study data, including electronic case report forms (eCRFs) linked with medical imaging and treatment planning data. Particularly noteworthy here is the combination of existing open-source projects, such as the Conquest [Conquest, 2021] server as PACS, OpenClinica [OpenClinica, 2021] for the entry and management of clinical data, and the Mainzelliste [Lablans et al., 2015] for data pseudonymization in order to realize the needed requirements. However, data processing is not addressed by this infrastructure. ePad [Rubin et al., 2019] is another open platform for medical imaging, specifically designed to simplify the viewing, annotation and quantitative analysis of cancer lesions. The system also consists of several modules which communicate with each other. Dcm4chee [Gunter Zeilinger, 2021] is used as the central data storage and a plug-in architecture also enables the integration of MATLAB [MATLAB, 2010] server-side modules for data processing. However, the core of this infrastructure is also more about the annotation than about processing of data using modern ML methods.

Where these projects were developed with an imaging focus, there are also projects driven by the data science community. These projects tend to focus less on clinical integration but more on the implementation of processing pipelines. NiftyNet [Gibson et al., 2018] is a well-known example of this, which is intended to standardize and simplify the handling of medical image data for data scientists. This is achieved by providing modules for various typical medical image processing tasks such as data loading, data augmentation, network architectures, loss functions, and evaluation metrics. NiftyNet is built on the Tensorflow framework and also uses this as the central DL engine. A similar concept is being adopted by Monai, which also aims to provide annotation, training and deployment of data and algorithms to the clinic. In contrast to NiftyNet, Monai is based on PyTorch as the core framework for DL tasks. In contrast to NiftyNet, Monai is based on PyTorch as the core framework for DL tasks. The MONAI Deploy [MONAI Deploy Working Group, 2021] Workflow Manager is also intended to enable more complicated processing pipelines and the Informatics Gateway provides connectivity to clinical systems through FHIR. However, due to the high dependence on PyTorch, data processing is not very flexible and data exploration and cohort specification are also not well addressed here.

In summary, there are already many established projects that can be utilized for the various requirements of the infrastructure envisioned in this work. Similar concepts with decentralized data processing based on textual statistical datasets have already been implemented and were able to successfully overcome the hurdles of high data

protection for medical data. Although some medical imaging platforms already exist, they are often commercial or closed-source and thus not applicable as an open research tool for the community. Open projects so far have on the one hand an imaging focus and offer good data integration, management, annotation, and visualization - but less for data processing. On the other hand, there are projects specialized in data processing, but offering little for the other aspects. Therefore, for the work presented here, existing open and established software projects with a high degree of specialization were integrated into a modular infrastructure, which allows the bundling of features and thus enables a comprehensive platform solution.

3 Methods

Based on the analysis of related work, pre-existing concepts and implementations of already available software projects for specific tasks were considered for the envisioned infrastructure. Following these findings, the main concept for Kaapana was developed and is presented in this methods chapter. Furthermore, stakeholders and typical use cases to be covered are also listed. The basic technologies and their architecture are described next, followed by implementation details of the various platform areas. The chapter concludes with explanations of the build system, the server installation, and functionality of the developed CI system.

3.1 Main Concept

The current situation with an increasing demand for high quality data on the one hand, and strict data protection requirements on the other hand, pose a great challenge for medical image processing. Since usually the location of data acquisition differs from the location where the data is finally processed, a data transfer is needed, which causes most of the described challenges. To address this bottleneck, a different approach was explored in this thesis:

"Let’s share the algorithms, not the data!"

The main idea is not to collect (and therefore transfer) data from multiple sites, but rather to share the data processing algorithms with the data owner’s and execute them locally within their own infrastructure. This reversal of the traditional approach solves many of the existing data protection concerns, since the data remains at the acquisition location at all times. A transfer and the associated problems with a distribution of personal data can be thus avoided.

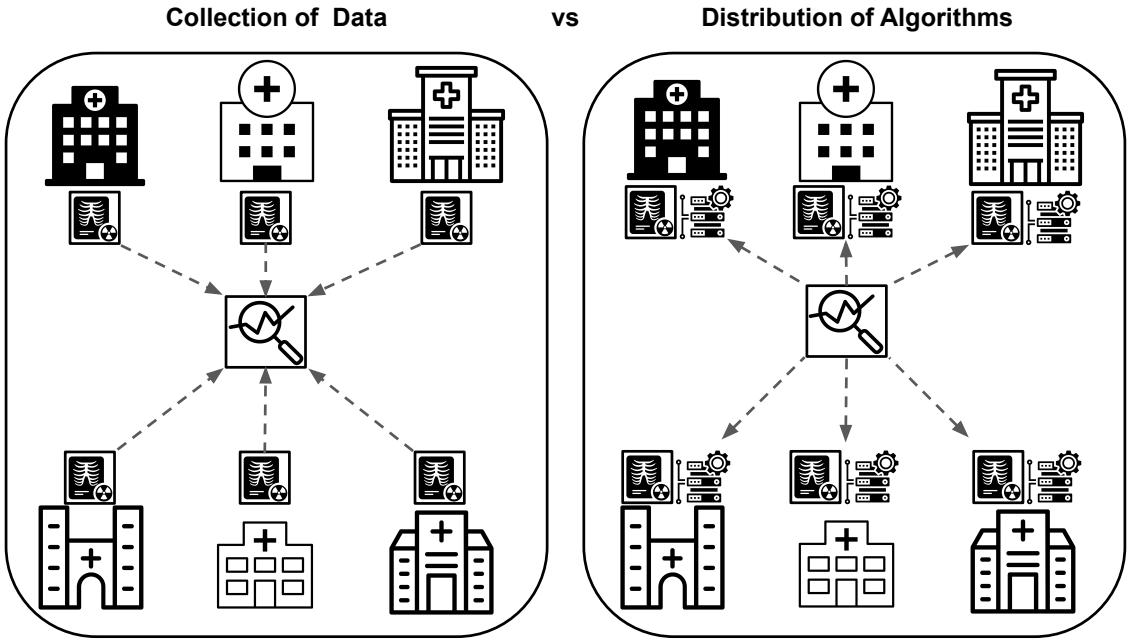

Figure 3.1: *Visualization of the "Let’s share the algorithms, not the data!" approach.*

3.1.1 Decentralized Infrastructure

To enable this, a decentralized digital infrastructure must be established, which allows clinics to install and execute recent research methods within their own IT infrastructure. This local infrastructure consists of two parts: The platform software, which enables the sharing and execution of data processing pipelines, and locally operated hardware servers, which run the platform software and provide the required computing power. The combination of server and software make up one instance of the network - the interaction of several such systems makes up the entire distributed infrastructure.

3.1.2 Integration into the Clinical IT Landscape

For a successful operation, a high acceptance by the clinical partners is crucial. This is achieved by a good adaptation to the existing IT systems and a seamless integration into existing clinical routines. Running the hardware right at the local sites allows physicians to send images to the analytics server straight from their workstations by providing all the necessary user interfaces (UIs) within the hospital's local network via a web browser. Due to the focus on medical imaging, the connection to the clinical PACS is particularly important and constitutes therefore the focus of this work.

3.1.3 Central Distribution Hub

The infrastructure is designed to run the local servers without any permanent external connections. This is important as clinical networks are subject to high security regulations and isolated systems offer advantages in this regard. However, for some processes like the initial installation, updates or the adding of new methods, a central hub for the distribution of the software is needed. This hub acts as an on-demand contact point and remote systems at the hospitals merely pull data when needed. No data upload of patient data is anticipated. By offering plugins, such a hub also enables an "app-store", through which new analysis pipelines can be provided and distributed.

3.1.4 Stakeholders

Various stakeholders are targeted for the application and development of Kaapana and the platforms that were created with it. Mainly two groups can be identified, each of them with a different expectation profile:

Medical / Study Stakeholders

This group includes the study hosts and clinicians (radiologists), whose interests are centered on making the study process faster and easier. Study hosts expect a widely available and standardized environment to facilitate study designs, as data provision and processing are already clarified. Moreover, hardware already optimized for data analysis can be utilized, which is normally not available in clinical environments. Also tasks related to data privacy or ethical questions should be straightforward, since only the data owners are handling the data. Clinicians, on the other hand, expect to have a more convenient way of using data from clinical routine for challenging research questions. By processing the data locally, they do not have to worry about tasks such as anonymization and can therefore collect the research data directly during their daily work. Here, especially the cohort compilation should be streamlined and the technical challenges should be abstracted.

Technical Stakeholders

The technical stakeholders are divided into three sub-groups: Data Scientists, technical staff and platform developers. For the data scientist, the opportunity to gain access to a broad and realistic pool of data is of great interest. Thanks to the network of trusted partners, their own methods can be evaluated at a variety of sites. In contrast to homogeneous, artificially compiled public data sets, the significance of a realistic clinical evaluation is clearly increased. Federated learning opens up new sources of data that make it possible to tackle highly specialized problems and to train models where it was previously difficult, due to a lack of accessible cases.

Since on-site servers have to be maintained by local technicians, efficient maintenance without a lot of special expertise is desirable. The staff expects similar procedures as they are already familiar from the other server systems they operate. And finally also the Kaapana developers are among the technical stakeholders. Such a framework can only be successful if it is accepted and supported by the Medical Image Computing (MIC) community in the long run. Here the expectations focus on public source code, welcoming contribution policies, and a flexible adaptability of the system.

3.1.5 Use Cases

In order to develop a better understanding of common tasks to be covered by such a framework and the derived platforms, several use cases and scenarios have been envisioned.

Use Case 1: Autonomous Execution

For the diagnosis, treatment or monitoring of medical conditions, it is often helpful to have additional information available. Since medical imaging can contain a lot of additional information, it would be helpful for studies to have access to this data. However, since the extraction of this information is technically challenging and complex, a system is needed that automatically performs these analyses and provides the extracted information in a standardized way. For this use case, the platform is used to autonomously detect the appropriate images for a given analysis to then automatically start the processing and provide the information acquired to the study supervisor and the physicians. For autonomous data processing, all images from a hospital or a group of scanners are automatically forwarded to the analysis server after acquisition. Using a rule-based decision system, the platform determines whether the incoming image is suitable for automated processing and if so, triggers the corresponding processing pipeline. The results of the analysis are then transmitted back to one of the clinical systems, where they can be accessed by the healthcare professionals. Typically, this is a research PACS that is used to manage the data, which is not for clinical use. Alternatively, the data can also be made accessible through the UIs of the platform itself. The central issue of this use case is the identification of the corresponding eligible images, since the rule-based decision system only works if the required information is available. Most algorithms define the accepted input data by the modality and the body part examined of the images. Where the modality is practically always accurately recorded in the corresponding DICOM tag of the metadata, the reliability of the information in other, less standardized metadata entries is often significantly worse.

Case Scenario 1.1: Existing Metadata
In this scenario, all the information needed for the execution decision is reliably available within the metadata. The platform extracts it accordingly and checks it against the decision system.

Case Scenario 1.2: Pre-analysis and Metadata Extraction
In the scenario where this information is not available, the incoming data must first be pre-analyzed so that the corresponding meta-information can be detected from the images itself. Furthermore, such an analysis can also be used for the verification of already existing metadata by comparing them.

Use Case 2: Integration of Data Analysis Algorithms

For the information extraction described in use case 1, methods are needed that automatically analyze the images and extract the desired information. For this purpose, existing and well performing methods from the research community are adopted and integrated into the platform by processing pipelines so that they can be easily applied. The challenge here is to offer a concept for the integration of these methods that is as universally applicable as possible and reflects the many different approaches in the form of data, programming languages, frameworks or hardware requirements used in the community. Here, classical central processing unit (CPU)-based processing techniques should be supported as well as the latest DL techniques, which rely heavily on GPU computing resources. The platform should remove as much of the integration effort as possible from the method developer by already providing typical processing steps such as file format conversion or other pre-processing steps. Provision of templates should also facilitate this last step of integration, so that any developer can accomplish this step without much knowledge of how the underlying platform works.

Use Case 3: Integration of Services and Desktop Applications

The goal of providing a genuine infrastructure platform, in the sense of a foundation that can be used for the provision of all kinds of other services, requires that third-party software can be integrated easily. Since Kaapana is web-based, services that are already designed for web browsers together with the use of the client-server model are the main focus here. However, not all applications to be provided meet this requirement, so desktop software should also be taken into account as far as this is possible. The nature of the services to be integrated can be very diverse: From research applications for the assessment of specific disease profiles to web-based development environments, all kinds of services are plausible. The following scenarios can be distinguished for the provisioning of services within Kaapana.

Case Scenario 3.1: Permanent Services
The service will either be installed during the platform installation or as an extension and will then run permanently. If a UI is needed, it is provided via web interface and can be accessed within the platform.

Case Scenario 3.2: On-Demand Services and Interactive Processing-Pipelines
The integrated service starts on demand only and provides its features during the execution of a processing pipeline. This scenario describes an interactive data processing, where a part of the pipeline is executed fully automatically and then halts temporarily to wait for a user interaction. Manual user interaction is enabled using an integrated service as described above, which also provides temporary UI access if required.

Use Case 4: Interactive Data Annotation

Another application of the platform is the generation of image annotations, which are useful for studies and the training of new ML-based processing techniques. The challenge of making these can be categorized into three aspects. The first difficulties are technical and concern finding, loading and opening the images of interest in software that allows the creation of annotations. Since this process is mainly used in research and not for daily clinical procedures, clinical systems such as the PACS often do not support the creation of such labels. Even if commercial solutions exist, they are often not available in the clinics due to the costs involved. Consequently, research software is used for the labeling, which can only be installed locally on a workstation, and thus the images must be manually selected and retrieved from the PACS.

The second challenge lies in the annotation itself, as often many of the three-dimensional layers of an image must be manually edited in order to produce the desired labels. The repetitive and monotonous work also creates the risk of low attention, which in turn increases the risk of mistakes. Finally, the third challenge involves the storing and managing of the generated information. Choosing non-standard or proprietary file formats, which can lead to a loss of information, for example, due to the lack of metadata inclusion, are not preferable in this context. In contrast, it is better to use open standardized formats such as DICOM, which allows joint management of annotations and images in the PACS. All three aspects should be addressed with the platform, leading to standardized annotations that can be managed along with the original data.

Use Case 5: Machine Learning Workflows

The last use case involves the training new ML models using the data available in the platform. This is of interest when there is not yet an automatic prediction method for a particular task, but sufficient annotated data is available to train a new model. It may also be the case that a pre-trained model is available, but does not work well using the locally available data. In this case, the model could be refined using local annotations and images, so that a specialization of the model to the local conditions is achieved. Also for training, it is important that annotations can be imported into the platform or that they can be generated from scratch as described in use case 4. There are two fundamentally different ways of running the training.

Case Scenario 5.1: Local Training
On the one hand, there is the strictly local training, which uses only data available within a single hospital. Here, no external connections are required and everything is executed on the clinic's in-house hardware. The model generated can subsequently be used for predictions and, if desired, can also be shared with other sites running a Kaapana-compatible platform instance.

Case Scenario 5.2: Federated Learning
If there are not enough training cases available at a single site to train a model, or if locally trained models do not generalize well, distributed training across multiple sites may be beneficial. Here, data from multiple sites can be utilized to train a model without having to share the data itself. There are many different approaches for federated training, which should in principle also be feasible with Kaapana.

3.1.6 Unified Execution Environment

A uniform software environment is key to successful multi-organizational collaboration. Interfaces, such as data provision, job scheduling or metadata access must be identical across platforms so that developers can rely on them - no matter if they work on a development or any of the production instances at the hospitals. This primarily affects the following aspects:

Data Formats

The standard for the communication and management of medical imaging information and related data is DICOM, which is used internationally in almost all medical facilities. Compatibility with this standard enables Kaapana to handle images from most manufacturers or scanner types. Besides transmission and the storage of images, DICOM Tags play a central role, since this metadata is used for data exploration and cohort definition. It is not only used for external communication, but is also the central medium within the platform and its internal services.

DICOM is mainly important for the medical and study stakeholders of the platform - but there are also the data scientists, which mainly use formats like NIfTI and Nrrd for the handling of medical imaging data. Although these formats are often considered easier to handle and process, they also have the disadvantage that meta information is often lost during the conversion. This problem is addressed by Kaapana by providing tools to temporarily convert DICOM to the commonly used formats while keeping track of all metadata for a later re-conversion. This allows the processing pipelines to request the format they were designed for, avoiding the need of dealing with DICOM for the data scientists. Since analysis results should be saved in standard-compliant

DICOM objects, the framework also provides the corresponding tools to generate such files. Important formats include DICOM Structured Reporting (DICOM SR), DICOM Parametric Map (DICOM PM) or DICOM Segmentation Objects (DICOM SEG), which can be used to store results like segmentations or classifications. The reduction of effort needed to port methods into the framework should strengthen the acceptance of the data science community and facilitate the integration of the latest scientific developments. Other data, such as trained models, are not specified any further due to the lack of established standards and can be freely chosen.

Data Access

The data of interest should be kept as a duplicate to the clinical records, where they can be retrieved for analysis from a separate research PACS. This system is the main storage system of the platform that manages and stores all DICOM compatible files. This redundant data storage is necessary to ensure unaffected operation of the clinical infrastructure. Within the platform, the data can be accessed via a DICOMweb-enabled [Genereaux et al., 2018] Representational State Transfer (REST) application programming interface (API) of the internal image archive. All non-DICOM data is managed and stored in an object-based storage (object store) [Factor et al., 2005]. This also provides data access via a REST API, but should only be used in case no other option is available. The framework already provides modules to simplify the search, storage and retrieval of data from the various components of the platform.

Data Selection and Cohort Definition

Since most processing techniques are highly specialized, they are dependent on the correct characteristics of input data. For example, it is not advisable to analyze an MRI of a brain with a model created for liver segmentation. Therefore, the selection of data with certain properties is required, before an analysis can be performed. This selection is realized within Kaapana by filtering metadata obtained from the DICOM headers. The data contains patient information as well as technical parameters from the acquisition of the scan. A search engine allows filtering by specific criteria through search queries defining a cohort, which can also be shared with partners or services within the platform.

Processing Pipeline Definition and Execution

Performing data analysis is Kaapana's core mission, which should not be seen as a single procedure, but rather as a pipeline including multiple processing steps. Starting from the original images, several pre-processing operations such as data conversion, normalization or image cropping are often necessary. Likewise, the final resulting

data often needs to be post-processed or forwarded to other systems. The framework provides standardized processing by predefining specifications and interfaces for both the pipeline sequence in the form of Directed Acyclic Graphs (DAGs) and for the individual processing steps. The fact that only the interfaces and not the way of implementation itself are specified leads to no restrictions with regard to the frameworks, libraries, programming languages, etc. to be used.

Within Kaapana, the execution of data analysis roughly follows the Extract, Transform, Load (ETL) procedure [Denney et al., 2016]. Here, the data to be processed is first retrieved from the source systems, then it is processed, and finally the results are uploaded back to the corresponding target systems. The processing of medical image data, of course, can be very different from the typical database operations, which are usually performed in ETL processes.

Extract: At the beginning of each processing pipeline, the data to be processed must be retrieved. This is achieved by using a query (see 3.1.6), which filters the data available to the desired input data. After the target data is identified, it is fetched from the internal archive and followed by the retrieval of referenced related images. This is useful, if the selected DICOM contains derived data, such as segmentations or image features associated with a base image, the annotations are based on. For example, the input for a ML is specified by a query targeting the ground truth labels - the framework then gets the corresponding referenced base images for the training automatically. Alternatively, also metadata or files from the object store can also be retrieved.

Transform: The transformation usually takes several parallel or consecutive processing steps. Here, pre-processing such as conversions, cropping or normalization are carried out first, while tools for the most common processing steps are already available. This is usually followed by the core pipeline task such as classification, object detection or image segmentation. Then the generated information is post-processed and, if possible, converted into a standardized form. Again, the framework already provides many of the typical operations, so that the main analysis method can be easily placed between the provided processing steps.

Load: The final step distributes the results back to the target systems. For DICOM compatible data, this is primarily the internal PACS - if desired, they can also be transferred directly to the clinical PACS due to standardization. In case of extracted metadata, it is updated and extended in the metadata database. All non-standard processing results are stored in the object store in a predefined way so that they can also be retrieved and associated with the source data.

Pipelines built accordingly can be installed very easily in Kaapana compatible systems. Special care was taken to balance simplification by abstraction with freedom for developers. The principle follows the concept that the framework should adapt to the method more than the other way around.

Data Analysis vs Services

Where the integration of analysis pipelines provides the environment for the inclusion of new analysis jobs, a second environment is required for the provisioning of services. The distinction between the two is well defined in the context of Kaapana. Where processing jobs are always started on demand only and always terminated after their job is done, services constantly provide their functionality within the platform. Services are software components that provide a certain range of features and often include a separate user interface. Thus, for instance, the internal research PACS, the metadata search engine or the integrated image viewer are all examples of services, whereas a file conversion, normalization or liver segmentation are all processing jobs within Kaapana. Also for services, defined interfaces are provided to integrate new components into the Kaapana infrastructure.

Extensions

For flexible platform design, Kaapana offers the possibility of dynamic expansion of the feature set thanks to software extensions. Essentially, all software components within Kaapana, such as processing pipelines or services can be deployed as extensions. A key point here is that they can be added dynamically at runtime to the platform, enabling an "app store" for deploying new analytics pipelines or services by and for the community.

3.1.7 Kaapana

The main vision of Kaapana is the creation of a standardized digital infrastructure, which is capable of supporting all kinds of projects in the context of medical image computing. As outlined in the scenarios, such projects can have very diverse requirement specifications, which is why not a single rigid platform was built, but rather a whole framework. Software platforms derived from Kaapana can be seen as instances of the framework, which do not necessarily have to use all the components offered. Thus, specific platforms can be generated that are adapted to the respective project needs. Building a platform with Kaapana involves the following steps:

1. Deploying the base-platform of the framework.

2. Selection of the necessary software components from the Kaapana software catalog to meet the project requirements.

3. The integration of new software components for the project requirements not yet covered.

Functional Units

The JIP, for example, is a concrete software platform generated using the Kaapana framework. This is achieved by providing many compatible software components that cover all kinds of tasks needed for the operation of such a platform. These components can be divided into separate functional units, responsible for a given range of core tasks:

SYSTEM: This layer contains all components needed to provide basic platform capabilities, which includes managing platform access, provisioning of services, communication between them or the support of dynamic extensions. The system layer creates the foundation every platform is built upon, which makes it part of the Kaapana core. The Kaapana SYSTEM can be considered to be like a runtime environment (RTE) of the platforms, which is also monitoring the platform's status by periodically collecting and evaluating status reports from its deployed services. The state of the host machine is also constantly checked by logging e.g. CPU, GPU, random-access memory (RAM) or network utilization, to react in the event of unexpected or faulty conditions by restarting services or triggering alarm messages.

BASE: This unit is designed to serve as a central UI. Although platforms are composed of many separate software components with independent UIs, the BASE allows them to be combined into a single UI in the form of a website.

STORE: This section is responsible for the management and storage of a variety of data types within a platform. The main components here are the PACS for DICOM data and the included object store for other files. Since the image viewer is tied directly to the data stored here, it is also part of it.

META: This unit is dedicated to the metadata within the framework. Besides storing and maintaining the information, there is also the possibility of a visual filtering and examination of the data.

FLOW: This Is responsible for all components related to data processing, in particular the pipeline execution environment and the collection of Operators representing common processing jobs.

Taken together, all of these functional units provide all of the core features needed by powerful software platforms to cover the wide range of application scenarios. For the basic implementation of Kaapana presented in this thesis, the JIP project has played a central role, as this was the initial platform for the framework and thus provided many of its requirements. The JIP is therefore also meant when the following text refers to a platform instead of the framework.

3.2 Technological Foundation

After defining the requirements and the concepts, the technologies for their implementation have to be selected. Even though the word "platform" is certainly used inflationarily for all kinds of software nowadays, Kaapana is supposed to be the basis for actual, genuine platforms in the more narrow sense. Bottcher defined a digital platform as “an operating environment which teams can build upon to deliver product features to customers more quickly, supported by reusable capabilities.” [Evan Bottcher, 2018], which matches the concept of a platform referred to in this thesis.

To build the framework, a suitable base technology had to be identified initially. Here, the modularity, flexible extensibility and simple maintenance were particularly important, as these are directly affected by the base technology. Also, the need to centrally host the platform at the hospital, rather than running one instance on each workstation, led to the decision to base the framework on current cloud computing technologies.

A cloud seems contradictory at first, as it is usually associated with services from big technology companies, which are located in data centers spread across continents and accessed via the Internet. This appears to be in direct conflict with the requirement of decentralized, isolated and on site operation of platforms in absence of any external connection. However, this can be solved by adopting the private cloud deployment model: “The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.” [Mell and Grance, 2011]

3.2.1 The Cloud Computing Stack and Architecture of Kaapana

Building a (private) cloud computing setup traditionally requires an infrastructure, a platform and the application layer [Mell and Grance, 2011]. Kaapana, adds two extra layers - a container provisioning layer for virtualisation and a function layer (serverless) for on-demand data processing [CNCF, 2021]. Figure 3.2 shows the arrangement of the layers and how Kaapana is designed upon them.

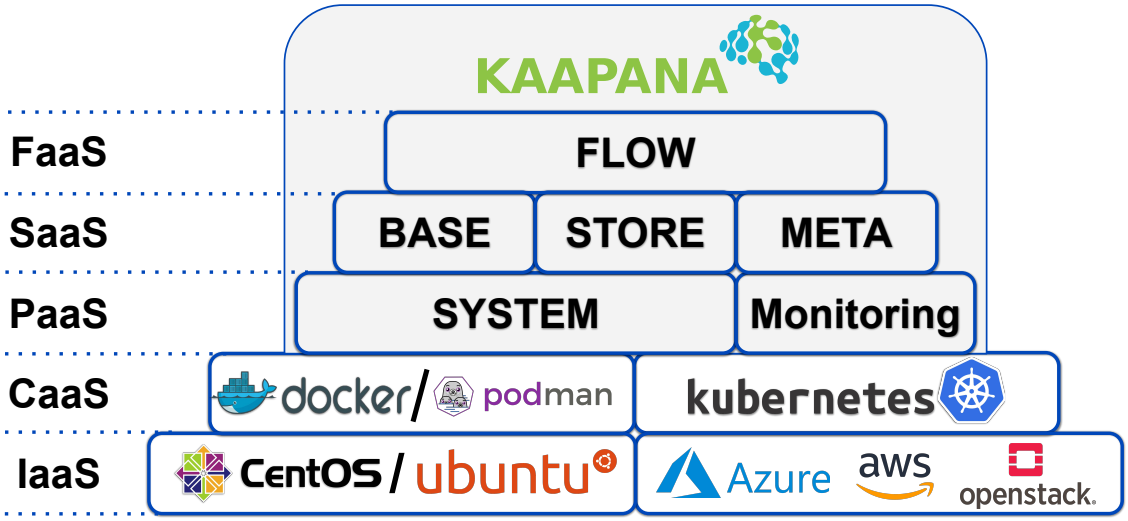

Figure 3.2: *Kaapana's technology stack with its different layers.*

Infrastructure-as-a-Service (IaaS) Layer

Kaapana supports on the lowest level both the common IaaS providers such as OpenStack [The OpenStack Foundation, 2021], Azure [Microsoft, 2021], GCP [Google, 2021] or Amazon Web Services (AWS) [Amazon, 2021b], but can also be deployed on-premise on a blank hardware server without any dependency on IaaS providers. While this requires considerable extra effort to enable non-experts to be able to set up and maintain such a plain hardware infrastructure, it also allows the needed private cloud deployment model. Besides instructions on how to install the operating system (Ubuntu and CentOS are supported), a custom installation script is offered to set up the basic infrastructure on top of the operating system (OS).

Containers-as-a-Service (CaaS) Layer

All components of Kaapana are not executed on the host system itself, but in containers supported by OS-level virtualization. The resulting encapsulation offers several advantages, as containerized platform and system components can be scaled, monitored and replaced very easily. Consequently, component updates can be made simply by replacing the corresponding containers. Since a platform requires a large number of containers for its operation, it is necessary to be able to monitor and control them centrally. For this purpose, Kubernetes (K8s) [Rensin, 2015] has been chosen as the container orchestration engine, which enables such functionalities.

Platform-as-a-Service (PaaS) Layer

The platform layer, as the name suggests, is where the Kaapana SYSTEM is located. It is based on the PaaS (or application platform as a service) principle and offers Interfaces for the provisioning of applications on top of the container infrastructure provided by the IaaS layer. The reverse proxy and the authentication system are also part of this layer, which thus provides everything needed for the provisioning of services. The design follows the principle of service-oriented architecture (SOA) for the provision of the described features [Perrey and Lycett, 2003]. This layer represents the base-platform of Kaapana, which is still "empty" and provides only basic platform functionality. All the basic functional units for service provisioning, job processing and UIs are in place, but domain adaptation is not yet included. Therefore, the base platform can be used as a foundation for many other projects - even outside the imaging domain.

Software-as-a-Service (SaaS) Layer

Initially, there is only the base-platform in Kaapana and this layer is empty. But depending on the project requirements to be addressed, services can be added. These software components can either be taken from Kaapana's service catalog or be integrated as custom services. The default configuration of the JIP already contains a set of applications needed to realize the outlined scenarios (see 3.1.5). At this level, the BASE, STORE and META units of Kaapana are located since they're represented by services on the platform.

Functions-as-a-Service (FaaS) Layer

For on-demand scheduling of processing jobs, another layer based on the FaaS principle [CNCF, 2021] has been added to the architecture. The difference between services running on the application layer and jobs (functions) running on the function layer is described in the "Data Analysis vs Services" section (see 3.1.6). These jobs are started on demand only and terminate automatically after completion, which enables high scalability by high parallelization and optimally utilizes the underlying host system. Kaapana's FLOW unit is powered by this layer.

3.3 Implementation of Kaapana

After the requirements, concepts and technology selection, this section focuses on the implementation of the Kaapana framework. Here, it is important to note that, as identified in the analysis of related work, many software implementations for required features were already available. Therefore, the implementation was based on established and successful open-source projects as far as possible. Many packages were collected, combined and modified to form new features for Kaapana, which was also released as open-source code. The goal here was to encourage contributions from other developers and thereby strive for a joint project and to give back to the MIC community.

3.3.1 Package Management

Given the demand for modularity and individual, interchangeable software components, there must be a central way to manage them. This is achieved by using Helm Charts [The Linux Foundation, 2021] for the package management, which is a widely-used standard for managing K8s deployments. Here, package collections are hierarchically structured and each package can contain sub-packages, allowing the construction of functional groups consisting of several software components. For Kaapana, the various functional units have been modeled as such packages, that can be added or omitted as needed. However, much more fine-grained decisions can be made if necessary, since all functional units are in themselves based on packages that can be chosen individually.

Each platform is defined by a main package, which defines the project's respective sub-package dependencies. This is also used to differentiate multiple platform versions and such allowing maintenance-routines, since Helm offers options for a standardized installation, update or removal of deployments. In this context, the ability of templating is also important, which is used in Kaapana to adapt and configure deployments to given circumstances and specifications. Helm Charts are stored, managed and distributed by the support for the Open Container Initiative (OCI) [OCI, 2021] standard, allowing charts to be wrapped by containers and thus managed with OCI-based registries.

3.3.2 The Base-Platform: SYSTEM

As described in the main concept of Kaapana, the SYSTEM enables it's base-platform. This is achieved by using K8s to define interfaces for the standardized provisioning of services and already including software components for monitoring, administration, ingress management, and user authentication. A high level of integration has been achieved by automatic pre-configuration, which ensures all components hosted in

Kaapana to be compatible with each other and do not require any manual configuration adjustments.

Integration of Web-Applications as Services

Since Kaapana is implemented as a cloud solution, its services are provided centrally via a server. Native software components for Kaapana are therefore web applications that have already been developed according to the client-server model. Even though this may not necessarily involve a web UI, the software itself should be deployed as a webserver and be accessible via network interfaces. Almost all services from Kaapana's software catalog have been integrated in this predefined way, as well as the system components also follow this principle. Three components are required to integrate a new service:

First, the application (if not already available) must be packaged into a container. It should be possible to configure all settings using environment variables, as this is the main configuration mechanism within Kaapana, which can be set externally by the framework if required. By assigning versions in the form of container tags, different development stages can be mapped. Containers can then be imported into the platform using **K8s deployments**, which also define the required configuration environment variables and storage provisioning in the form of volume mounts. The definition of expected resource utilization and limits ensure the appropriate allocations on the host system. Periodic healthiness checks via API requests can detect unresponsive and thus non-functioning components and restart them respectively. It is also possible to wait for the availability of other services, such as databases, before starting the application.

The second ingredient specifies network traffic, which involves specifying the appropriate ports and all network interfaces that will be available within the platform. **K8s services** are used to register these access points with the internal Domain Name System (DNS) and to handle all corresponding container traffic. An additional Ingress configuration is only necessary if a web-based user interface is to be offered. The configuration of the automatically provided reverse proxy, which is responsible for processing and routing all Hypertext Transfer Protocol Secure (HTTPS) traffic, is done via K8s Ingress objects.

The last ingredient involves the packaging of components using **Helm Charts**, which are wrapping the K8s definitions for all deployments, service and ingress objects into a complete package. If multiple deployments are required for a component, for example, due to multiple web servers or additional database servers, they can be bundled during packaging. Any manual configuration should be available via templating to

allow for customization during installation. These packages can then be added to the software catalog and either delivered as a default platform dependency or deployed as a dynamic extension.

Integration of Desktop Software Components

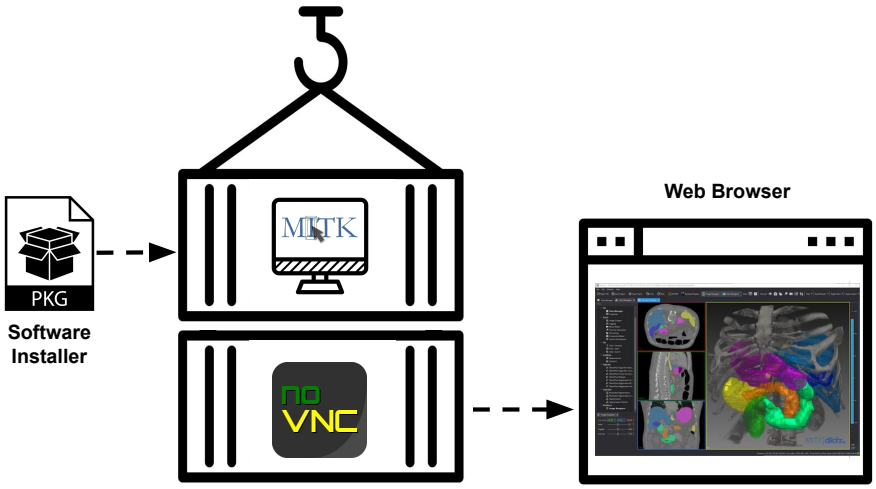

Figure 3.3: *Schematic representation of desktop application containerization.*

Since not all applications of interest for the integration into Kaapana are already web-based, another method was developed for the integration of desktop applications. Instead of a conventional container for web applications, a specialized base-container is provided which hosts a desktop via Virtual Network Computing (VNC) [Joel Martin, 2021]. This way, the desired desktop application can be installed inside the container, and its UI is then offered by a remote desktop via a web front-end, like a web application. Figure 3.3 illustrates the integration of the desktop image processing software MITK Workbench, which can be accessed via a web browser using the VNC based method.

Ingress Routing and Identity/Access Management (IAM)

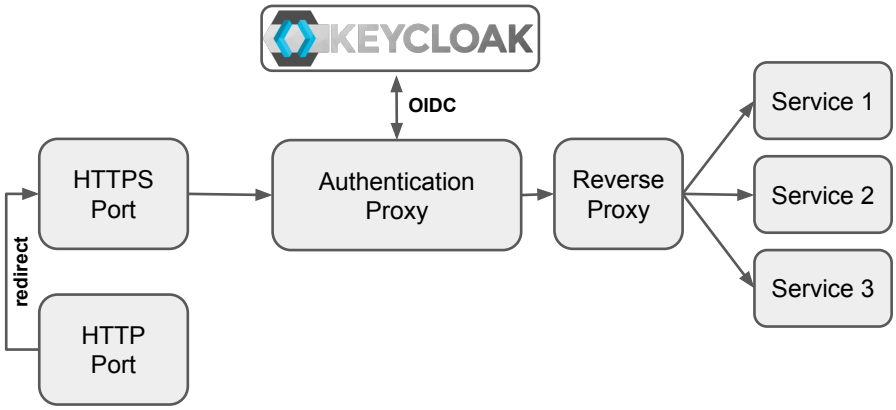

Figure 3.4: *Schematic of the HTTPS traffic routing of Kaapana.*

Since UIs from all hosted services have to be delivered via web interfaces and HTTPS, they need to be routed in order to deliver multiple interfaces from a single point of entry. Kaapana addresses this by using a central reverse proxy that connects each service via Uniform Resource Locator (URL) sub-paths. This will also be used for load-balancing within the platform, in case of increased load requiring multiple instances of a single service. The reverse proxy Traefik [Ludovic Fernandez, 2021] was chosen, which has some advantages over other solutions in the context of Kaapana. First, there is a dashboard, which provides up-to-date routing metrics, service reachability and status information via a web UI. This is helpful for managing the platform as it can quickly identify network traffic routing issues. Most importantly, however, is Traefik's ability to dynamically detect and adapt to configuration changes. Kaapana uses this automatic service detection to dynamically deploy extension UIs, as they can be integrated and removed at runtime. Traefik is embedded in the platform via K8s Ingress, allowing all components to be configured via K8s objects.

Some of the services use their own authentication systems, which implies that a user has to log in multiple times and users are redundantly managed in different locations. Since this is not ideal, with Keycloak [Keycloak, 2021] a central OpenID Connect (OIDC) enabled authentication provider has been integrated within Kaapana, which can be used across all components. This single sign-on (SSO) strategy is not only more convenient for users, but also facilitates the connection to existing IAMs at the hospitals by supporting the Lightweight Directory Access Protocol (LDAP) or Kerberos as widely used standards. Last but not least, central authentication management also enhances security and enables role-based access control (RBAC) by assigning roles.

Traefik in its free open -ource version does currently not support direct user authentication via OIDC, which led to the integration of another component into the HTTPS processing chain. HTTPS requests to the server are first checked by an authentication proxy [louketo, 2021] for the presence of a valid authentication token. If this is the case, the request is forwarded directly to Traefik and thus to the central distribution system - if not, the user is redirected to the login. Louketo not only verifies a user's successful authentication but also their assigned role, which can be used to restrict access to services. Currently, two user roles are implemented (administrator and user), but these can be customized and extended at any time.

Monitoring System

Figure 3.5: *Screenshot of the server monitoring dashboard based on Grafana.*

The design of Kaapana offers a lot of capabilities, but also involves a significant level of complexity. To streamline platform operations, maintenance and administration, a centralized monitoring toolchain was integrated, which is used to get an overview of current of system and component states. The event monitoring and alerting system consisting of Prometheus, Alertmanager [Prometheus, 2021] and Grafana [Grafana Labs, 2021] is a common toolchain used in cloud computing. Prometheus regularly collects metrics from all supported components, which can then be evaluated. The specification of rules controls the triggering of alarms, so that, for example, notifications can be issued in the event of a resource shortage. In addition to the platform components, the host system is also monitored, so that CPU, GPU, RAM or the network load are logged regularly. Figure 3.5 shows how all this information can be visualized

and accessed via dashboards in Grafana. For the monitoring of active deployments, a K8s dashboard [kubernetes, 2021] has been integrated, which allows to view all logs of the containers or to restart services. Since this provides administrative access to the platform, this dashboard has been restricted to the administrator role only. The information gathered is also used for the scheduling control of the data processing. Before a processing job is started, Prometheus is first checked for the current system utilization. Since the typical loads of the jobs should be known, it can then be decided whether a job can be executed on the system or not.

3.3.3 User Interface: BASE

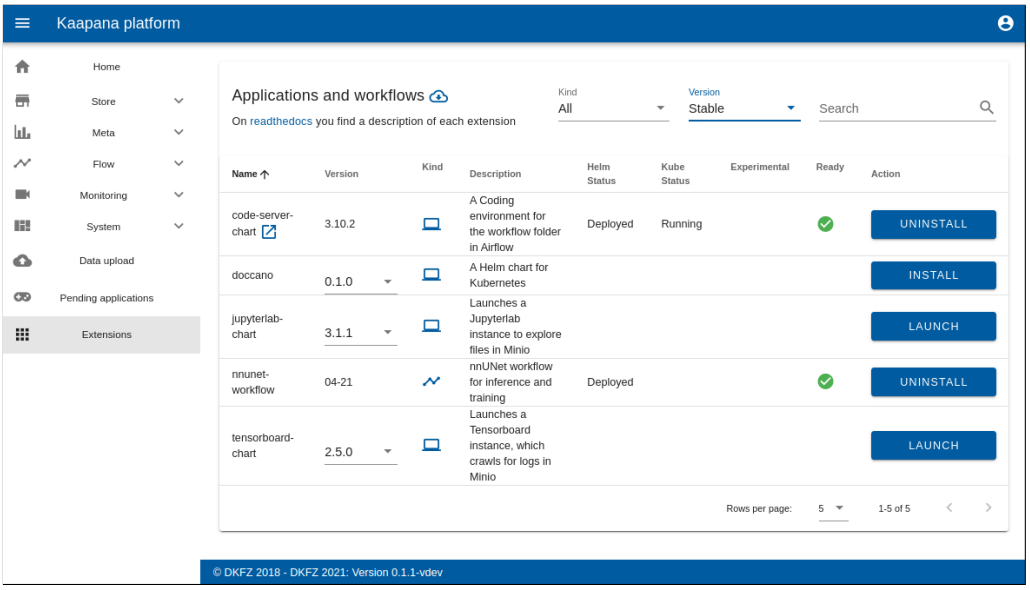

Figure 3.6: *Screenshot of the Extensions section from the landing page.*

As Kaapana's concept intends, the BASE provides a unified UI. For this purpose, a web application based on Vue.js [Evan You, 2021] has been developed, which provides the body for all independent service UIs. With the help of iFrames, the individual services, which are actually accessible via URL sub-paths, are merged into single-page application (SPA). Service presents is dynamically checked and corresponding menu items are activated or deactivated accordingly. It is structured in such a way that a common generic base is available from which project-specific landing pages can be derived. The customizations primarily focus on branding, using project-specific designs such as texts and logos.

Extensions

The BASE also includes the extension management (see figure 3.6), which includes a user interface within the landing page as well as a management backend. In principle, all components of Kaapana can be provided as optional extensions, which means that not only services but also processing pipelines can be installed via extensions. The UI allows searching and filtering by specific package names and distinguishing between stable and under development packages by labels. In order to announce the current deployment status, the backend reports updated information periodically. The extensions mechanism has been realized by the implementation of an API on top of the integrated package management, which allows dynamic installation or removal of packages.

3.3.4 Primary Data Management: STORE & META

Kaapana as a versatile processing framework needs a central resource: Data. The concept, which is fundamentally based on cloud architecture, results in most of the hosted services being either stateless or use external databases to manage their data via network interfaces. Here, the great advantage lies in the ability to deploy services across different host systems, as the data can be retrieved and stored via interfaces across servers accordingly. In the interests of modularization, all stateful components are deployed with their own data management systems. That means that databases used by services are not shared, resulting in increased robustness and flexibility as components can be more easily replaced and updated. Data managed for data processing is kept in central data repositories and distribution systems, which are used by all services to retrieve or store data. Within Kaapana, **DICOM**, **meta-** and **other** data are distinguished for data management:

DICOM

As described in the Data Format section of the Standardized Environment of chapter 3.1.6, DICOM is the worldwide established standard for the communication and storage of medical imaging data, especially in hospitals. For this reason, it was chosen as Kaapana's central data format, which is used wherever possible for the storage and communication of data. Besides raw voxel data, DICOM also comes with valuable metadata, including patient information, technical parameters and even data lineage information. With the help of dedicated Information Object Definitions (IOD) [NEMA, 2021b], not only image information but also other data can be represented with DICOM. The Segmentation IOD, Parametric Map IOD, Encapsulated Document IOD and DICOM SR IOD [NEMA, 2021d] are particularly important for Kaapana, as they are used to represent processing results. This not only allows the results to be

managed together with the images, but also to be shared with other external clinical systems if desired. The default guideline for the development and application of Kaapana is: Whatever can be covered by DICOM should also be represented with it.

Data Collection: For the integration into the local IT environment, an easy data transfer from clinical workflows into the platform is key. For this purpose, Kaapana offers both a C-STORE [NEMA, 2021b] and a DICOMweb-based DICOM receiver [Genereaux et al., 2018] that can be defined as targets within the clinical PACS. This should enable clinicians to forward images to the platform from their daily workflow while using the PACS. Images can also be automatically and rule-based forwarded from the scanners to the server. This allows, for example, all images with a specific study description to be forwarded to the analysis. The Called Application Entity (AE) Title plays a special role here, as it can be selected freely and serves as a data identifier within the platform. If, for example, the AE-title "Study 1" has been selected, the corresponding data can be found and selected within the platform by this dataset name. The setup of the DICOM receiver has been realized on the basis of CTP by the Radiological Society of North America (RSNA). Two import pipelines accepting C-STORE or DICOMweb, respectively, were established, exporting the files to the server's file system.

The next step in the CTP pipeline involves a custom plugin that notifies the FLOW system of an image's arrival by triggering the incoming DICOM pipeline, which takes care of the distribution of files within the platform. All destinations to receive the DICOM files can be configured within this pipeline - the default configuration specifies the internal PACS only, but this can be extended to include any additional targets. Afterwards the processing is taken over by the FLOW system. Besides this direct transmission via the PACS, compressed DICOM files can also be uploaded as a ZIP file via the UI of the landing page.

Data Storage: As in clinical systems, Kaapana also hosts a PACS for the management of DICOM data. With dcm4che [Gunter Zeilinger, 2021], the default platform configuration contains a mature and comprehensive image archive. The high level of standardization of these systems allows an easy replacement of this component if a different implementation is desired. The interfaces expected in Kaapana for the platform integration are limited to C-STORE for sending images to the archive and DICOMweb for retrieving data for processing, which also enables the integration of externally already existing research PACS within the hospitals.

Data Retrieval: Kaapana identifies and manages data using the typical DICOM properties of patient, study, series and object. According to DICOM, studies are not

meant in the sense of clinical trials, but rather examination sessions, which can contain various individual examinations. Studies contain series which correspond to what is usually called a single data record, or more commonly "image", in the context of medical imaging. A series, in turn, contains one or more objects, which correspond, for example, to the individual layers in a three-dimensional image. Since series are mostly used for data processing, and the individual subcomponents (such as individual layers) are processed rather rarely, Kaapana's data model is centrally based on the series as a unit. The request of a single data record is characterized by the unique identifier (UID) of the corresponding DICOM series, which results in fetching the complete image including all layers. This concept is also going to be very important in the next chapter when defining metadata and cohorts.

Since within a distributed system the processing can theoretically be executed on a different machine than the server running the archive, Kaapana never processes the original data of the PACS, but always creates temporary duplicates created via network transmission. This is achieved by giving all components access to the DICOMweb interface, by which specific series can be queried and downloaded. Typically, data exploration and the associated cohort definition are used to identify the desired series, which are then loaded directly from the infernal archive via Hypertext Transfer Protocol (HTTP) requests.

Visualisation: The integration of the OHIF Viewer [Urban et al., 2017, Ziegler et al., 2020] allows Kaapana to visualize stored DICOM data right in the browser (see figure 3.7). The display of a series can be initiated either by selecting it in a study overview of the viewer or from META dashboards. This has been achieved by implementing the Integrating the Healthcare Enterprise (IHE) Radiology Technical Framework Supplement Invoke Image Display (IID) [Committee, 2016] in the OHIF Viewer, which allows a specific DICOM Study to be accessed by a standardized URL. The HTTP-based connection of the internal PACS via DICOMweb allows the viewer to be treated as any other service within Kaapana, since the UI can just fetch the data from the web browser. Apart from displaying three-dimensional scans such as CT or MRI, annotations like segmentations can also be displayed as an overlay, which allows this viewer to be used to inspect and present processing results. This viewer even handles more exotic formats like Encapsulated Document IOD [NEMA, 2021c], which can be used to display DICOM compatible PDF documents.

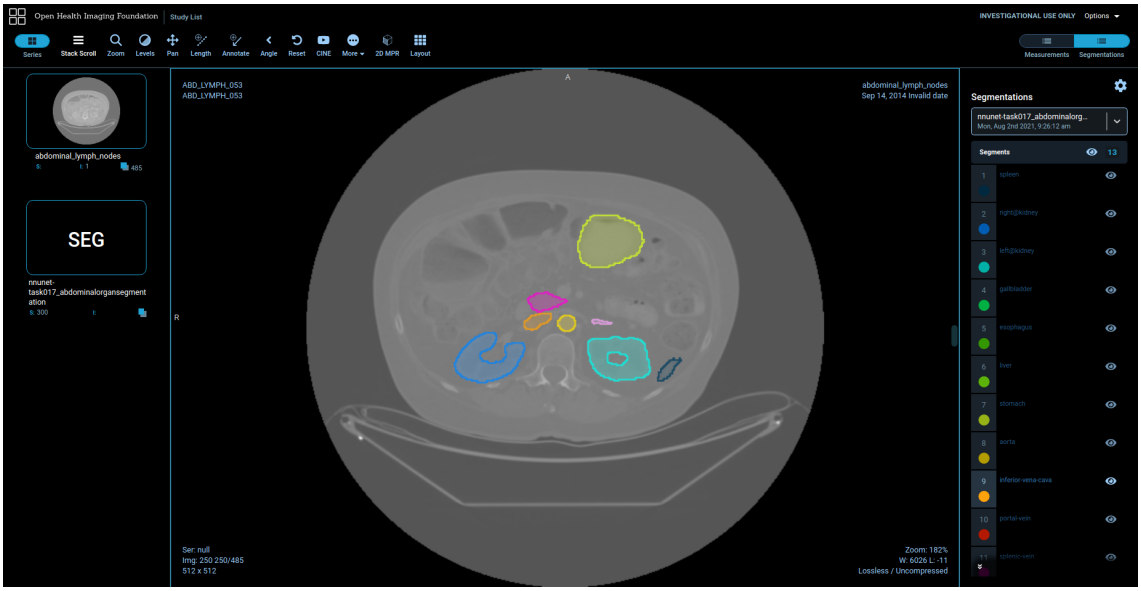

Figure 3.7: *Screenshot of the integrated OHIF Viewer.*

Metadata

The second main category of data in Kaapana is metadata, which is characterized by providing additional information about existing data entries. Kaapana's DICOM-centric data model indicates that the metadata model is also centrally based on DICOM headers and built on a series basis. The metadata system was designed as a different view of the data, which means that it should not contain any additional information about the data stored in the PACS, but only visualize the textual data it contains and make the entries filterable. The metadata is mainly used to explore the existing datasets and find out if there is suitable data for a given analysis or the training of a new model. This is achieved by feeding the data into a search engine, which allows full-text searching in the tags.

Data Collection: The metadata system contains the header information contained in the DICOM data, which gets extracted after the arrival of a new series. As described in the DICOM - Data Collection section, after the arrival of a new series, data processing will be initiated via FLOW. Here, first of all, a pipeline for the extraction of the metadata gets automatically triggered, which consists of three processing steps:

1. Using DCMTK's dcm2json [OFFIS, 2021b], the DICOM files are converted into a JSON file according to the DICOM JavaScript Object Notation (JSON) model [NEMA, 2021c]. This file contains not only the raw image binary data but also

deeply nested DICOM header information, which cannot be processed this way by the search engine and therefore has to be cleaned.

2. Cleaning up the JSON and adaption to the search engine. Here, the data is prepared by first removing the binary information and private DICOM tags. Then, for readability, the hexadecimal-based tag identifiers are extended by their clear names. Although the standard specifies data types of different tags by DICOM Value Representation (VR), data in the real world often does not adhere to the specifications. Especially pseudonymization or anonymization procedures often result in incorrect data, which will be identified and solved here.

   Dates and times take on a special role, as a variety of different formats are permitted and common. During import, the existing format is initially identified and then transferred into a uniform format. Due to different time zones during data acquisition, the time shift is also eliminated by converting the data to Coordinated Universal Time (UTC). Elasticsearch (ES) [Elastic, 2021], which is the Apache Lucene [Apache, 2021b] based NoSQL search engine used, allows full-text searches, however, they become more and more time-consuming with high nesting, so that this should be avoided. In the last step, the nesting will be reduced by transferring the data types from a nested style of the DICOM JSON model into a key extension. The result is a JSON file where the keys contain the DICOM tag, the clear names and the data type - data types are verified to be correct and all dates, times and datetimes are in the same format and time zone. For instance, the entry for the DICOM tag (0008,002A) - AcquisitionDateTime looks like this: "0008002A AcquisitionDateTime_datetime": "2020-08-13 12:16:56.000000".

3. Transfer of the metadata to ES. The last processing step involves uploading the JSON file to the database. Before a new file is uploaded, it is checked whether data already exists for the corresponding series - and if so, the data is expanded and updated.

Data Storage and Initialization: For an efficient search, ES builds an internal index, which is based on all occurring keys, each defining fixed data types. For a flexible initialization and extension of the index, a dynamic indexing was implemented, which extracts the corresponding data type from the key name and includes it accordingly in the index. Since the system is series-based, the series UID is used as the Primary Key of the database and the acquisition time of the series is set as the time of data capture. Since the index dynamically adapts to new entries, initialization is not mandatory, as it would build up piece by piece over time. However, this would lead to an initial faulty presentation for the dashboards, since the referenced index entries do not exist after

installation. Therefore, an initialization job was introduced, which initializes the ES index and also checks the DICOM processing pipeline by sending a custom DICOM file internally to the receiver. This file contains dummy header information for most of the existing tags, which leads to the initialization of the ES index and performing a self-test at the same time, ensuring that all components of the DICOM processing pipeline are operational.

Visualisation, Cohort Definition and Data Retrieval: As described in the requirements, data exploration and cohort specification are key features for the framework. The visualization of metadata has two goals: First, it should give an overview of the data available on the server, which is important to quickly develop new ideas for potential analyses, or to evaluate whether an existing analysis or training on the given data would be reasonable. For visualization, Kibana [Elastic, 2021] has been deployed, which supports the creation of dashboards including tables and graphs on top of ES. Along with the graphical summarization of the available series, customizable dashboards also allow for click-based filtering of data by including or excluding fields in the corresponding graphs. Filters created that way are jointly translated into a search query, which is then used to define the desired cohort, which represents the input data for a processing pipeline. So DICOM metadata is used to select data that fits to a given analysis or training. A typical example would be an algorithm that predicts certain anatomies within CT images of the abdomen. In this case, the DICOM tag (0018,0015) - "BodyPartExamined" would be looked at to filter just those images that indicate the abdomen as a body part here. The second DICOM tag of interest would be (0008,0060) - "Modality", as it specifies the modality of the scans. Adding the filter "Modality is equal to CT" results in the desired data consisting of abdominal CT images. Kaapana's default configuration includes three dashboards, each with a distinct focus:

1. **General Purpose Dashboard:** This view is intended to provide a general view of the data, by including both patient and technical related information. Here, for example, the distribution of genders, age or weight of patients of the data on the system can be identified. In addition, all DICOM specific information such as study or series names is available as well as technical parameters such as when the scan was acquired or which scanner was used. Even the application of contrast agents can be displayed and filtered in diagrams, if the data is available.

2. **Segmentation Dashboard:** This view has a focus on segmentations, as they are one of the most common annotations of image data. Here, the data is already prefiltered to DICOM SEG, and specific properties such as the segmentation labels present or the algorithm names and versions used to create the segmentation are shown. This dashboard serves primarily for selecting ground truth data for the training of new models.

3. **Model Dashboard:** The final default dashboard focuses on machine learning models represented as DICOM. The tasks of this view are related to installing, uninstalling and forwarding of models available on the platform. The information presented refers to algorithms, versions or parameters used for training the models.

Thanks to Kibana's flexible customization capabilities, users can develop, add and share their own dashboards at any time, which allows new custom views of the data with other focuses. Now that the filters needed to define a cohort can be created, they need to be transformed into a Lucene-style search request to be passed to the FLOW system to be used as input data. Here, it is important to note that in the context of Kaapana, a search query defines a cohort at a given time. Since the data on the system can change over time, the result of the query may also change. This is desirable because it theoretically allows input data to be defined across systems and sites, which would not work if explicit series UIDs were used. Furthermore, in the future, the available data for any analysis or a training can be constantly monitored in order to be able to notify the platform user in case of a changed situation (for example, the availability of sufficient data for a training).

For the extraction of the Lucene search query and the triggering of a processing pipeline including the transmission of the cohort, a new plugin for Kibana was developed, which can be integrated as visualization in dashboards. The visualization consists of a simple dropdown list, which lists all pipelines installed on the platform, and a button to start the corresponding selected pipeline. List entries are always fetched from the FLOW system via an API request when the dashboard is loaded, and therefore automatically adapted to changing system configurations. Furthermore, once the start button has been activated, it is possible to adjust parameters that are accepted by the pipeline. This is realized via a popup dialog, which contains a dynamically generated form based on JSON Schema [Pezoa et al., 2016]. This allows the relevant parameters to be defined by the processing pipeline, which is then automatically displayed when the process gets triggered. The extracted cohort query together with the parameters are then transmitted to the FLOW system and the triggering is finally completed with a status notification. Within the pipeline, the query is evaluated using ES. The response includes all series UID matching the given criteria and will automatically be downloaded from the internal PACS via the DICOMweb interface. Further details of this mechanism can be found in the following chapter: "Data Processing and Pipeline Execution".

Other Data

So far, the data described could be standardized and assigned to a specific category - but there are also cases that do not fit any of the existing categories, which are then classified as "other data". This applies, for example, to image data that is not available as DICOM, but rather as NIfTI or Nrrd, as well as to all kinds of processing results which are neither suitable for metadata due to their volume or structure, nor does DICOM conversion appear to make sense. Due to the unspecified properties of this data pool, it is simply managed as files located in an object storage. However, the great flexibility of this data category has the disadvantage that hardly any standardized interfaces can be offered to simplify management, searching or visualization.

Data Storage: To store and manage this category of data, MinIO [MinIO, 2021] as a high performance, K8s-native object storage has been integrated into the framework. The flexible management of files up to a size of 5 terabytes (TB) and compatibility with the widely used Amazon Simple Storage Service (S3) API [Amazon, 2021a] are particularly interesting features which facilitate interaction with other components. A user interface is also provided, which also allows manual uploading or downloading of data via a browser. Links can be generated to access files with limited validity, which facilitates data sharing. The interconnection to Kaapana's SSO service allows the adoption of users and roles for specific permissions within the file management.

Data Collection and Retrieval: The data import can be realized in two ways. First, they can simply be uploaded via the web interface so that they are available to the system when requested. The second way is more common, as the data is automatically uploaded by the FLOW system during a processing pipeline. For sending new data and receiving existing data, FLOW offers jobs that utilize the S3 API. The caching system implemented in FLOW is also based on MinIO, as results to be cached are automatically stored in corresponding buckets within the object storage. Again, integration via network interfaces allows easy replacement of the service if desired, or linkage to existing systems in the respective clinical infrastructure.

3.3.5 Data Processing and Pipeline Execution: FLOW

The main idea behind the implementation of the data processing was to enable the integration of new methods as easily and with as few adjustments as possible. Therefore the top layer of Kaapana's technology stack supports FaaS, or serverless computing. Why is such a layer necessary and what distinguishes it from the underlying SaaS layer? As described in the "Data Analysis vs Services" section (see 3.1.6) of the ain concept, the distinction between services and processing jobs is the reason why this extension is necessary. Where the SaaS layer enables all services that run permanently

as an application within the platform, FaaS provides the ability to launch on-demand jobs. The FLOW section of Kaapana allows the definition and execution of data processing pipelines and associated jobs, the basic principles of which have already been explained in the Pipeline Definition and Execution section of the Standardized Environment. Here, the actual implementation of these concepts is presented.

Workflow Management System

In order to make data processing possible, a workflow management system is required to handle the mentioned tasks. Due to the great flexibility, a complete workflow management system was chosen for the underlying framework instead of relying on existing FaaS frameworks, as these already impose great limitations on job implementation and design. Furthermore, these systems are usually not designed to handle large amounts of data, which simply overwhelms their concepts given the file sizes and computational demands that are typical in medical data processing. For example, often the input data is submitted via a single HTTP request, which works for a small text snippet as input, but just won't work for medical images. Kaapana therefore uses Apache Airflow [Apache, 2021a] as workflow management platform, which has been extended by APIs to offer the corresponding FaaS functionality. Especially the REST interface for the interaction with META and the extension for the communication with the K8s cluster described within the Operators section are important here. For easy integration of all customizations added for Kaapana a dedicated Airflow plugin has been developed. Airflow also provides a web-based user interface to view and manage the installed pipelines. By displaying running processes in real time, analyses can be tracked and corresponding logs of the individual processing steps can be easily inspected.

Processing Pipelines: Directed Acyclic Graphs (DAGs)

Since Kaapana's workflow system is based on Airflow, its terminologies are adopted here for the description of the components. In this context, processing pipelines are synonymously referred to as workflows and are realized with the help of DAGs. A DAG is characterized by the step-by-step processing of individual jobs, which are called Operators in Airflow. Practically, a workflow consists of a simple Python script, which imports the corresponding Operators as objects and orders them in the desired sequence. They are not only used for scientific data analysis in Kaapana but also for platform tasks such as metadata extraction or deletion of data records. This has led to a distinction between service and data processing workflows, which are both managed within the same system. DAGs can be started either time-based as a cronjob, event-based or by manual triggering, which offers flexible application for various scenarios. Most data processing jobs are triggered manually whereas service workflows tend to

be more event or time based. DAGs usually follow the ETL procedure, consisting of three parts: Data import, data transformation and data export. Compared to other FaaS systems, FLOW does not have a fixed specification of how these tasks have to be executed, as they are all handled by Operators and can therefore be easily replaced, adapted or omitted entirely. However, for common operations, such as downloading or uploading images to the PACS, Operators are already provided with the plugin. This also enables the integration of new data sources or simultaneous retrieval from multiple systems. For complex workflows it is also possible to create nested workflows by using so-called SubDAGs or by dynamically triggering DAGs during runtime.
The step-by-step sequential processing requires the data to be passed on accordingly. To keep the principle comprehensible, a file-based approach was chosen, where the output data of a given Operator serves as input data for the following one. In this way, the only interfaces to an Operator's DAG are a directory containing the input data and a directory to which the results of the processing should be saved. The corresponding folders are then automatically provided to the Operator at runtime and treated as temporary files through the filesystem. Since files can be used to represent arbitrary data structures, the system should be able to be used for all kinds of scenarios. The disadvantage of this is that Operators do not receive a uniform input, which could ensure compatibility across all Operators. Since this is unrealistic anyway due to the large variety of processing schemes, this point can be neglected. Consequently, the developer of a DAG needs to ensure the compatibility of the selected Operators.
Since workflows are simple Python scripts which are dynamically loaded into Airflow at runtime, new pipelines can be installed by simply copying the relevant files into the platform's DAG directory. For a comfortable and uncomplicated installation and removal of DAGs via the UI, the files are also wrapped into a container, which can be handled by the extension mechanism, which also provides the regular versioning capabilities due to the utilization of the general package management. For each platform, the DAGs to be added to the system during installation and the DAGs to be available via optional extensions can be defined individually.

Processing Jobs: Operators

The breakdown of the analysis procedures into separable stages leads to individual processing steps, which are referred to as Operators in Airflow. The advantages of this approach in contrast to the implementation of all stages in one large process are manifold. Most important is reusability and the accompanying provision of processing steps for other workflows. A task that has been implemented once can thus be easily used as a module in other workflows in the future. As a result, a large collection of Operators can be created in the long run, which offer solutions for all kinds of tasks. The second benefit is the traceability of the execution, since it can always be

determined at a glance which processing step is currently running, or which part of the analysis has failed. The provision of Operator-specific logs can significantly simplify troubleshooting. Another important point is the enhanced scalability of this approach. By dividing large tasks into small subtasks, they can be executed more efficiently on the host system. Because of the dedicated subtasks, the resource utilization can be estimated better and thus the system utilization can be estimated more accurately. By parallelizing many small subtasks, which can even be distributed to different servers if necessary, the system can be easily adapted to growing demands by adding new nodes.

An important factor for the free distribution of workload is the encapsulation of Operators. Just as with the platform's services, this is realized by using container technology. Thanks to their flexibility, almost all algorithms based on a wide variety of frameworks can be packaged, which also avoids the complexity that normally arises from the use of countless software dependencies such as libraries, etc. in this domain. Containers can be considered as standalone execution environments for the algorithms which can be exported and shared as complete packages. This prevents, for instance, two methods using different versions of the same library from interfering with each other, which in practice often leads to unreliability of algorithm behavior. Every processing step in workflows leads to the launch of a container within the platform. To enable these containers to be monitored and managed within the K8s-cluster as well, Airflow was connected to the CaaS cluster using the K8s API. The implementation of a *KaapanaBaseOperator* gives developers easy access for the development of new Operators, since the associated complexity is hidden by abstraction. The *BaseOperator* serves as a base class from which new Operators inherit, and thus automatically gain access to the interfaces to Airflow, which are outlined below:

Airflow interaction: In order to control an Operator within a DAG, lifecycle parameters can be specified, e.g., linking execution to conditions (such as successful completion of all previous jobs, etc.) or forcing termination by specifying timeouts and workload limits. Furthermore, specifying the expected workload also enables control of the scheduling system. Data coming in and out of the Operators is also managed at the *BaseOperator* by assigning specific targets in the temporary file system to each Operator in each DAG instance (also called DAG-run).

Container-based Operators: In order for Airflow to use containers to run Operators, the CaaS layer was tied in with the K8s Python Client [Kubernetes, 2021] and associated API. Besides controlling, this also enables monitoring of the corresponding container by periodically retrieving the execution logs, which are then automatically passed on to Airflow's own logging system, providing a central UI with access to the logs of all Operators in a workflow. The connection to the K8s administration also

has the benefit of being able to view and analyze all running containers in the K8s dashboard as well. The specified resources are not only used within Airflow, but also anchored to the container itself. This way, consumption is also registered at the platform level, which ensures that sufficient resources are left over for the operation of the core services.

Processing tools often include configuration options which can affect how the process is executed. In order to pass on these parameters, they must also be passed into the container, where they can be picked up by the algorithms. The specification of these parameters can be done in two ways. Either they are specified during DAG development, or at the start of processing, as described in the metadata visualization. These parameters are then passed via the trigger REST API to the DAG and thus to the included Operators. The *BaseOperator* then takes care of passing them into the containers by ensuring that the parameters set are available as environment variables within the container at runtime, so parameters can simply be retrieved and evaluated by the algorithms at startup. Besides the parameters, the input and output data are crucial for the data processing within the container. The access is granted by mounting the temporary files into the filesystem of the container. Again, environment variables are automatically defined, which tell the algorithm where exactly to find the input data and where to store the results within its environment. Typically, an algorithm then scans the input folder for specific data first and starts processing afterwards. Apart from the automatic provision of input and output folders, stored models for the inference of machine learning algorithms can also be automatically mounted, which allows the code within the container to be separated from the actual model and thus enables more flexible handling of differently trained models. For an easy and efficient integration of new methods, several templates are offered, which cover the typical container procedures for the processing of the data input, output and parameters.

It is also possible to provide multiple input Operators if more than one data sources are needed. In case a GPU is needed, it can also be provided via the *BaseOperator*. If several GPUs are available in the system, the scheduling system selects a suitable unit automatically and assigns it to the corresponding container. In the simplest scenario, all that is required for the integration of a processing container is to specify the container image and assign the corresponding credentials for the registry it is stored on. Once the template was used, everything else should be organized automatically.

Local Operators: Container-based Operators are well suited to accomplish large computations, but processing tasks are not always so sophisticated. Basic text processing such as extracting metadata from DICOM files is not demanding and the runtime of the operation takes only a few milliseconds. In these scenarios, starting the containers

within the infrastructure takes longer than the computation itself, so another category of Operators has been introduced to execute such small workloads right in the Airflow container. By providing another base class in the form of the *KaapanaPythonBaseOperator*, simple Python-based scripts can be executed immediately within the Airflow scheduler. This is very useful, for example, for a very large import where a great number of images need to be processed quickly to extract the metadata. However, the described advantages of the containers do not apply here. These small jobs must be programmed in Python and all dependencies must be pre-installed in the Airflow container. The biggest disadvantage lies in the lack of scalability, since the entire computational load of all local jobs is handled in a single container.

Application Operators: The last category of base Operators are the application Operators, which are used to implement workflows involving a manual element during execution. A common scenario is described in the "Interactive Data Annotation" use case, where an automatic pre-segmentation of an image is checked and corrected with the help of a manual processing step. Here, the behavior differs by halting the execution of the pipeline until the manual interaction of a user has been finished. To interact with intermediate pipeline results, a UI is typically needed to provide the user with access to the data. For this purpose, a new service is temporarily started in the platform, which offers the desired features, and is terminated again after the work is done. The user gets a new list entry within the “pending application” section of the landing page, which allows access to the waiting service via the web-browser. For the integration, a special *KaapanaApplicationOperator* has been developed, which, in addition to the normal container Operator features, also provides the ability to create K8s Services and Ingress objects that allow access to the UIs. Dynamic instance addressing can be enabled, which allows the simultaneous execution of multiple such services instances and life-cycle management ensures that all created containers and services are removed from the system even in case something goes wrong.

Important Default Workflow Processing Tools

Since most workflows follow a similar procedure, standard Operators have been developed to enable the core ETL functions. The Operators presented here are the most important default processing steps provided by the framework.

Input data preparation: In most cases, processing starts by fetching the requested image data. To provide a universal entry point, the so-called *LocalGetInputDataOperator* was developed, which supports all available input data definitions and provides the corresponding images as DICOM files. Input data can be requested in two ways:

1. **Elasticsearch Lucene query:** Here, the cohort is defined via Lucene queries, which result in concrete series through the request to ES, which are subsequently downloaded from the archive. They are typically defined via META dashboards and passed directly to the DAGs - but can also be hard-coded for specific DAGs so that, for example, all available CT images on the platform are always fetched during execution.

2. **DICOM series UIDs:** The alternative uses a list of predefined series UIDs, which are also downloaded from the PACS. This is used, for instance, if automatic processing needs to be started for a specific series, or if another DAG needs to follow for particular series during an analysis.

This Operator is used as a starting point for the following processing steps. The communication with the PACS was developed using the DICOMweb Client [Herrmann et al., 2018]. In case multiple related input files are needed, the *LocalGetRefseriesOperator* has been developed. DICOM references origin series, which derived annotations are based on, in the metadata. A typical application are DICOM Segmentation Objects (SEG) objects, which only contain the segmentation, but not the image information itself. However, since processing usually requires both, this Operator can be used to identify and download the image associated with the segmentation. Another use case would be to identify multiple related MRI protocols to be used as a combined input for processing. Similarly, PET also requires finding the source images. It is also important that expected but not found images are reported.

File format converters: Since most algorithms and tools in data science do not handle DICOM as an input format, providing robust conversion capabilities is important. The most widely used formats for non-clinical medical image computing are NIfTI and Nrrd. The *DcmConverterOperator* offers the possibility to convert DICOM files into one of these two formats. This has been implemented by utilizing the MITK file converter, which through years of development can very reliably convert all sorts of vendor specific formats. Integrated heuristics can also be used to reliably detect and discard faulty series, which is for instance the case for missing layers in a three-dimensional volume. Similarly the *DcmSeg2ItkOperator* serves to convert DICOM SEG objects into the NIfTI format. If a multilabel segmentation is present, it is also split into individual files so that these can be processed independently. The setting of filters also allows the extraction of specific labels from multi-label files. For the reverse conversion from NIfTI to DICOM SEG the *Itk2DcmSegOperator* is available. The implementation was realized using the dcmqi library [Andrey Fedorov, 2021], which is part of the Quantitative Image Informatics for Cancer Research (QIICR) project [Fedorov et al., 2016].

Textual processing results: For the conversion of text based processing results, which are available as JSON, there is the *Json2DcmSROperator*, which creates a DICOM Structured Report according to template TID1500 [NEMA, 2021e]. This functionality was also implemented using dcmqi. Another method to store and present textual or graphical results in a DICOM compatible way is to provide them as PDF files. DCMTK's dcm2pdf tool is used to create Encapsulated Document IOD compatible DICOM files, which can be managed in the PACS as well as displayed in the OHIF viewer.

DICOM wrapping for binary files: In case no other conversion is available, but the result should still be DICOM compatible, the *Bin2DcmOperator* has been developed. DCMTK's xml2dcm tool has been used to encode arbitrary binary files into DICOM. This Operator also provides the reverse functionality, by extracting the files again. However, since it is a custom encoding, these files can only be transferred, stored, and managed via DICOM, and other external tools do not have access to the included files.

Transfer of DICOM files: For the storage of the results, the DICOM files must be transferable to the target systems. The *DcmSendOperator* can be used to transfer these files via network connection to the internal PACS or even external clinical systems. It is also possible to exchange files between two platform instances using this method. This Operator is based on DCTK's dcmsend utility [OFFIS, 2021c].

Object storage support: For all results that cannot or should not be stored in the PACS, the *LocalMinioOperator* is available, which can store and query files of all formats in the internal object storage MinIo [MinIO, 2021].

Cleanup of temporary files: Since after the successful completion of workflows all relevant data should be stored in the associated target systems, the *LocalWorkflow-CleanerOperator* ensures that all temporary intermediate processing data is removed from the file system.

Integration Procedure for New Analysis Methods

The integration of a new analysis method follows three steps:

1. **Development of the algorithm:** First, the new method needs to be developed. For this it is advisable to work on the normal workstation of the developer. Normally, no special requirements have to be met during the development, except that the algorithm should be executable on UNIX-based OS and, if a UI is desired, it should be provided via web interfaces based on the client-server model.

2. **Containerization of the code using the templates provided:** In the second step, the developed code is packaged in containers. This is a common practice and already widely practiced. Adjustments are required for reading of input data and the writing of the result files - however, the provided template scripts should streamline this process.

3. **Integration into Kaapana by creation of a new DAG and Operators:** The last step consists of the actual integration into the framework. First, a dedicated Operator for the container has to be developed, for which templates are also available and which should be done with just a few lines of Python code. After that, the desired workflow must be defined by creating a new DAG. This involves selecting the appropriate Operators from the catalog that enable the provision of the desired file formats. Then the new Operator is included in the processing. Then the Operators for result handling are selected, which completes the integration process. Optionally, a DAG Helm Chart can be created to make the pipeline available as an extension.

Of course, this procedure describes only the most basic use case. Other, more complex workflows can also be implemented - but may then require several new Operators or adaptations to existing processes. As described in the results, this approach has been used several times both internally and by external partners to successfully integrate analytics into the framework.

Scheduler

One property of Operators is the varying computational demands. To run them efficiently on the host system(s), a scheduler is needed to coordinate the execution of jobs and ensure that there are still enough resources available for the platform's services.

For this purpose, a scheduler has been implemented, which monitors the consumption of the CPU, RAM, GPU and disk space for container-based jobs. The estimation takes place on three levels: First, the mechanisms provided by Airflow are used to make an initial estimation. These are mainly based on the idea of pools that provide slots and jobs consuming them. So Operators will be scheduled as long as the assigned pool still has enough available slots for the task. Since these pools are arbitrary, and initially do not represent any real resources, they are repurposed within Kaapana. Here, three standard pools are created, which correspond to the real resources of the host system. First, the CPU cores are identified, and each core is assigned a slot in the CPU pool. The second pool represents the RAM of the system, where each slot corresponds to one megabyte of RAM, which is not already used by the deployed services. And finally,

each existing GPU gets its own pool as well with the corresponding megabytes of RAM as slots. This way an Operator can be assigned to a pool, which consumes a specified number of slots. Unfortunately, it is only possible to specify one pool per Operator, which limits this specification to the most important resource of the job. Typically, these bottlenecks are the GPU or, if this is not needed, the RAM utilization. Airflow therefore estimates utilization by the pools and takes care of the basic scheduling itself. However, since not all kinds of resources can be considered in this process, and this estimation is based strictly on given estimates of developers, further steps are needed.

In the second step, which is executed after the approval by Airflow, all expected resource consumptions are compared with the states in the K8s cluster. Prometheus provides the actual system state and real readings as opposed to airflow estimates. Both layers are needed because of the delay between the triggering of a job and the actual load in the system during job execution. Consequently, if only the measured loads would be relied on, far too many jobs would be triggered due to the delay. K8s monitoring warnings, such as disk pressure, are also monitored at this level, and if warnings are present, execution of the job is prevented. In case multiple GPUs are available, the decision on which card the job should be executed is also made here. Several jobs can also be executed on a single GPU, which leads to significantly increased efficiency and allows large cards to be kept available for cases with high GPU demands.

The last level of surveillance involves the assignment of limits, which immediately terminates the job in the event that the specified requirements are exceeded. The K8s resource management takes all jobs and services running on the platform into account and prioritizes them if necessary. This prioritizes the operation of services so that the entire platform does not fail due to too many jobs being started.

3.4 Build System, Setup and Continuous Integration (CI)

At the time of writing, the framework contains about ninety containers and almost as many packages. This makes it very complex and cumbersome to build all the components and keep them up to date. To facilitate this process, a build system was developed, which can then also be used for the CI to regularly check changes to the code base, and to provide the current state of development. At the end of this section, the installation of the platforms will also be described.

3.4.1 Build System

The build system serves to collect all components of the framework, check them for formal errors, build them and finally transfer them to the target registry. The process begins by collecting all Helm Charts in the repository, which are checked by linting for formal issues related to the construction of Helm Charts as well as for compliance with the K8s standards.

All dependencies are then resolved and verified that all defined packages and versions actually exist within the repository. This also determines the order of the build process by building the packages with the least dependencies first. Thus, all Charts are built one by one until the final platform packages, containing all dependencies, are built. By using the OCI standard, the Helm Charts can also be packaged as containers and managed in registries. For each platform, a container is generated that defines the entire structure, all components and versions. During this process, all container images used in the packages are also registered in order to later match them with the containers present in the repository. In case images or versions are used which are not part of the current branch, an error is issued. This is important because, given the great complexity with so many components and versions, troubleshooting misbehavior quickly becomes unmanageable. In this way, it can be ensured that the components used in platforms also correspond to the versions that are currently available in the repository. At the end of package handling, all platform containers get pushed to the defined target system in the form of a container registry.

The second stage takes care of the containers. First, all containers in the repository are collected and checked for formal issues. Since containers inherit from each other, it needs to be assured that containers which are used as base images are built first. This also results in a build order, where it is also verified that base images are also present in the repository if not specified otherwise. The introduction of so-called local containers, which are only used as base images and are therefore not used by components, prevents them from being pushed to the registry. During the build

process of the containers, the output is checked for issues and the corresponding logs get extracted. At the end, all generated containers are pushed into the registry and a summary of all events is presented. Using multi-stage containers and uniform base-images keeps the memory footprint small and speeds up execution. Comprehensive caching during the container build-process ensures that once the framework has been built, only the adjusted parts will be rebuilt, which reduces build-time dramatically.

3.4.2 Server Installation

In order to establish a decentralized infrastructure, the software stack must be installable locally on dedicated servers. Since clinics do not always have experts for cloud computing software available to set up the servers, the installation process must be streamlined as much as possible.

Two separate stages are required to get a server up and running. First, all base dependencies for hosting K8s must be installed. Microk8s [Konstantinos Tsakalozos, 2021] is used here because it is very easy to install and maintain. As operating systems Ubuntu Server and CentOS are supported, since Ubuntu is often favored by developers and CentOS by clinical IT. For this purpose, a script has been developed which supports both operating systems simultaneously and installs and configures the required software. After this server installation step a single node K8s cluster is running on the server and Helm is ready to start deployment. This foundation is independent of the platform deployments and only needs to be set up once.

The second stage takes care of the deployment of the platform. Here, the first steps of streamlining were already accomplished by introducing package management, which automates the deployment and update of a platform, including all dependencies. Templating in the Charts enables all components to be configured centrally during deployment. When combined with the initialization jobs, this allows the provision of an out-of-the-box operational platform where components are already configured to interact well with each other. For deployment management on the server, another OS-independent script is provided. This allows the specification of the basic server settings like the file system location for storing data and ports to be used for the UI and DICOM receiver. Likewise, platform-related settings such as container registry, the platform's Chart name or the registry credentials can be made.

When the script is executed, the system first checks whether a deployment of this platform already exists. If this is the case, options for an upgrade, re-installation or uninstallation are offered. Otherwise, the deployment process is initiated, which prompts whether GPU support is desired, the target domain of the system, and

which version to install. After this is completed, the corresponding platform chart is downloaded via the container registry and deployed on the K8s cluster. It instructs the system which containers are needed and to download them from the source registry. For the monitoring of the platform startup, a command is displayed, which lists an overview of all components and their current conditions. Depending on the connection of the server to the registry, this process is completed in a few minutes and the user interface of the platform can be visited using a web browser.

The script also supports the integration of valid SSL certificates and the pre-fetching of the containers needed for the platform extensions. This ensures that they are immediately available instead of having to be downloaded during installation. It is also possible to set up the machine to launch offline without any connection to external systems. This is especially useful for isolated environments where the platform can be installed using connectivity once and then run in isolation without connectivity.

3.4.3 Continuous Integration (CI)

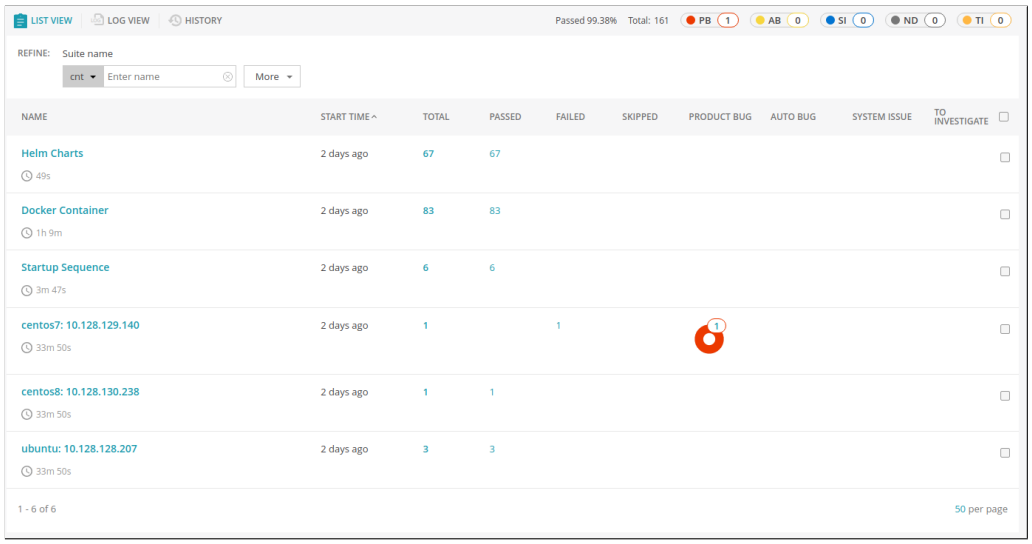

Figure 3.8: *Screenshot of the CI Dashboard.*

To constantly check the current code base, the entire framework needs to be built, installed, and tested regularly. This is achieved by continuous integration, which automatically executes and reports these procedures. Similar to the build system, there was no software available yet that could be used to perform these tasks for the way Kaapana is built, so it had to be designed and developed first.

The following tasks need to be addressed by the CI:

1. The repository should be checked out and built regularly.
2. Dedicated servers running various versions of Ubuntu Server and CentOS should be commissioned and the basic server requirements installed.
3. Different platforms (built in step 1) should be deployed on these servers.
4. Basic functions should be tested to verify the availability of the services.
5. Logs of all activities should be captured and visualized in a Dashboard.

For the execution and control of the CI processes, an independent server is used. Each night, the repository is fetched from the mirror, all cached containers are deleted and the build process for the framework gets started. If an error occurs during the build, the supposedly responsible developer is identified via Git and informed about the issue via email. For the automation of these processes Ansible [Ansible, 2021] was used, including playbooks for both the tasks to be executed locally on the CI server as well as those to be executed on the remote servers. Afterwards, all necessary servers with relevant operating systems will be launched. Since this involves frequently setting up and dismantling several servers, the system was implemented on the basis of OpenStack, which was already available as a IaaS provider at the German Cancer Research Center (DKFZ). After the servers are up, they are prepared for Kaapana by running the provided script for the installation of the server dependencies. Here, the quiet-mode is selected, which uses all default settings and does not require any user input. After the script is successfully completed, the servers are rebooted and ready for the deployment of a platform. For testing different platforms, the deployment servers are kept, since the operating systems and the setup to be tested is the same.

In the next step, the first of the defined platforms is deployed to all remote servers. During startup, all containers must first be downloaded from the registry. Since it consequently takes some time for the platform to boot up, the CI server monitors and logs this process on all servers in parallel. After the system reports all containers running and all initialization jobs completed, it proceeds with testing. These currently consist mainly of UI tests, opening the user interface on all servers, testing logins, and verifying the accessibility of all included components. Since the platform already includes self-tests such as sending sample images at startup, this is already verified when successfully completed. All reports are forwarded to a dashboard implemented using ReportPortal [Andrei Varabyeu, 2021] so that events and issues can be conveniently reviewed (see figure 3.8). For manual on-demand triggering of the process, a small web server was also developed that allows to select specific branches of the repository via a HTTP API and have them executed by the CI pipeline.

4 Results

For the evaluation of the concepts and the resulting implementation of the Kaapana framework, both the requirements and the scenarios are reviewed and evaluated based on actual projects. Furthermore, the partner sites participating in the projects are essential for the evaluation of an infrastructure. The JIP has been developed as a practical implementation of a platform utilizing the Kaapana framework. It has been distributed and utilized across two major consortia of university hospitals in Germany. Through real-world execution of clinical use cases, both the advantages and disadvantages of the current implementation are analyzed and as a result, possible future improvements were derived. Since Kaapana is an open source framework [Scherer et al., 2020a], it has already been considered by other groups and projects as a basis for their projects.

4.1 Evaluation of Requirements

First, the core areas of the requirements were evaluated. Here, the integrability, data and algorithmic accessibility, data exploration and cohort specification and finally the maintainability are presented. This involves analyzing and evaluating the coverage of these requirements by presenting concrete features that are used to realize the specific scenarios and use cases.

4.1.1 Integrability, Maintenance and Future-Proofness

The core of the main concept is based on the establishment of decentralized servers, which are operated locally at the sites. This means that local operation and integration into the existing IT environment is crucial. The first advantage here is that only one server needs to be operated per site. Due to the central on-site provisioning of all features and browser-based user interfaces, no additional software needs to be installed and maintained on the staff's workstations. With the support of CentOS, an operating system is supported, which is considered as stable and maintenance-friendly by many clinical IT departments, which allows the servers to be managed similarly to other Linux-powered servers in the clinics IT environment. The provisioning of scripts for the server setup and updates facilitates a quick start-up, which does not require any technology-specific prior knowledge of cloud infrastructure. By integrating package management with Helm, deployments can be handled very efficiently by installing, uninstalling and updating complete packages and the use of stand-alone containers enables easy replacement of components. By only requiring an external connection during installation, the server can be operated in isolation, which is important for applications within clinical networks.

By choosing out-of-domain, industrial technologies as the base layers, future-proofing is also increased, since these technologies have a very large user base and receive support from the large tech companies. This increases the likelihood that the open source projects utilized will continue to be developed and maintained in the future. Additionally, the advantages of modern cloud technology, such as scalability or modularity, can be used to deploy it in medical scenarios without relying on external servers. This would even make it possible to combine all clinical servers into one large cluster, if this becomes desirable in the future. Data and computing resources could thus be shared and utilized together on demand in a controlled environment.

4.1.2 Data Accessibility

Two factors are critical to data management within Kaapana. First, all processing must be done on redundant copies of the clinical data to avoid any interruption to the clinical operation and to ensure that no relevant patient diagnostic and treatment information can be altered. This is achieved by ensuring that all data to be analyzed needs to be copied from the clinical systems to the Kaapana server before any processing takes place. By integrating a DICOM receiver, the platform supports the clinical standard for the transmission of imaging data, so that it can be directly attached to the clinical systems. Physicians can therefore transmit the images for analysis via the user interface of the on-site clinical PACS, in the same way as they are used to from other clinical workflows. By providing an in-platform research PACS, such as dcm4chee, all images are redundantly stored internally and delivered to the analysis pipelines when needed. Since data scientists are often rather inexperienced with the DICOM file format, the included converter allows easy data supply in formats such as NIfTI or Nrrd. The same applies to the provisioning of annotations, which are also supported in the typical clinical and data science formats.

The second factor addresses the generation of hospital-compatible results that can also be transferred back into clinical systems if needed. The fact that DICOM is not only consumed by the platform, but also produced, means that most of the processing results can also be managed in the internal PACS together with the source data. A uniform conversion of results to DICOM will promote standardization within the community. Offering tools that automate and simplify these conversions as much as possible can significantly reduce the obstacles of method integration for developers. Operators such as the *Json2DcmSROperator*, *Pdf2DcmOperator*, *Bin2DcmOperator* or the *Itk2DcmSegOperator* already support simple DICOM generation for many analysis result types. In this context, DICOM also has the advantage of providing more information about analysis procedures through its metadata. Consequently, segmentations represented by DICOM not only contain a reference to the source image they have been created on, but also information about the algorithm used, which improves the follow-up analysis and management of analysis results. This standardized reference handling within the annotations serves also to automatically provide required data during the processing. For example, if radiomics features are to be calculated for an existing mask, only the relevant masks need to be selected. The workflow automatically retrieves the corresponding source images from the internal PACS and provides them for the analysis. In case the annotations should be available within the clinical PACS, they can also be automatically assigned to the correct patients and studies thanks to their references. Unfortunately, there is not a standardized DICOM representation for every kind of output category. For example, the local training of methods results

in models in form of large binary files, which currently cannot be represented by any DICOM extension. The supplied *Bin2DcmOperator* allows binary files to be encoded in DICOM, which allows standardized management and transfer of data, but the content can only be used in the context of Kaapana.

To restrict access to the platform and its data, a central SSO system has been integrated. By being connected to the clinical LDAP, the hospital's existing user and role management can be used for controlling access to the platform. All components of the framework have been successfully covered with the central authentication system, which strengthens the user experience as well as improves overall security. Besides general access to the platform, role-specific access to specific components can also be configured, so that, for example, the K8s dashboard is only available to system administrators. However, role-specific access to certain study datasets is currently not possible. As soon as a user gains access to the internal PACS, all data stored in the platform can also be accessed.

4.1.3 Algorithmic Accessibility

For successful data processing, not only the data but also the corresponding algorithms must be available within the framework. Two concepts facilitate and streamline the integration of methods into the framework. First, containers provides a way to package the algorithms independently so that individual algorithms can use their own programming languages, ML frameworks or software dependencies in general without any interference with other system components.

The second part here is the introduction of a FaaS layer, which enables pipeline processing of individual processing steps. This modularity enables a high degree of reusability of individual processing tasks, so that the developer only needs to take care of the actual method integration while other tasks, such as conversions or data organization, are handled by existing components. The platform's job scheduler takes care of distributing and scheduling job execution on its own, so the developers don't have to worry about hardware utilization and also ensures that resources are utilized as efficiently as possible to leave enough left over for the operation of the platform itself. The methods can then be offered by the extension system, which uses a container registry to provide both the pipelines as DAGs as well as the containers for processing. For the JIP, an independent central hub was established at DKFZ, which provides an OCI compatible container registry through Harbor [Harbor Authors, 2021] to deliver the platform and processing containers as well as the installation packages.

4.1.4 Data Exploration and Cohort Specification

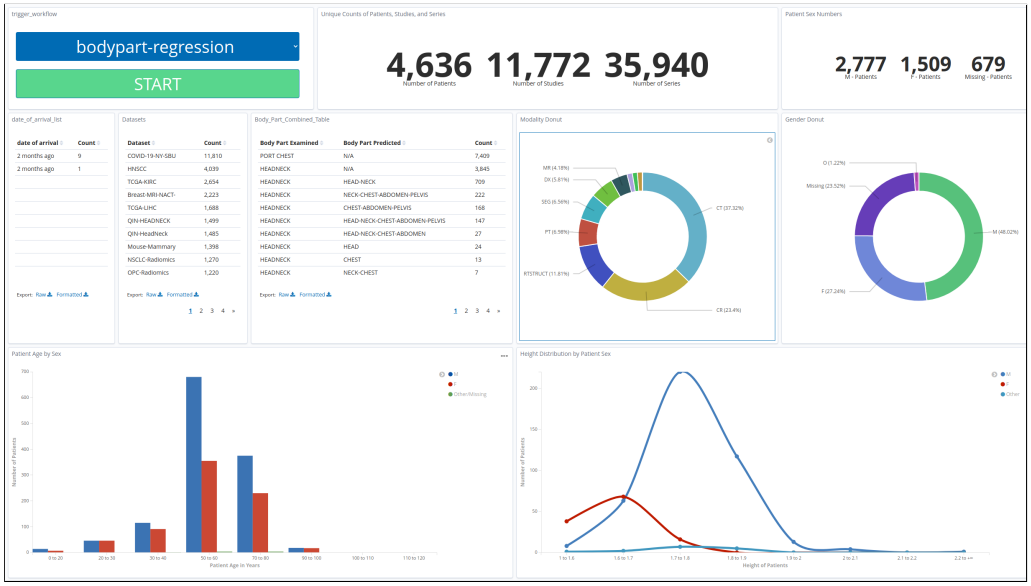

Figure 4.1: *Screenshot of the General Purpose Dashboard of META.*

The execution of any analysis requires a data selection first, so that the correct input data can be filtered out of the existing data collections. In Kaapana, various metadata dashboards have been introduced to provide an overview of the current inventory and furthermore to filter cohorts based on specific application requirements. The dashboards shown here have been populated with several collections from TCIA so that an impression of the various capabilities can be obtained. It is important to note that the data is anonymized, which means that some of the metadata entries have missing fields. In the case of real clinical records, it would also be possible to present more patient-specific insights, which has been omitted here due to privacy concerns.

Figure 4.1 shows the initial section of the META General Purpose Dashboard (MGPD), which provides a broad view based on the number of patients, studies and series of DICOM data currently stored on the platform. A pie chart here shows the distribution of DICOM modalities in the system, which allows narrowing down the data by a click-based selection of the target variable. Below are exemplary curves of gender-specific age, height, and weight distributions. The following technical data can provide information about which scanners produced certain data and at which time. Therefore, for example, it can be quickly analyzed on which days of the week and at which times specific machines are particularly utilized. It is also interesting that contrast

agent dosage can be included in the metadata, which when plotted along with patient weight, quickly reveals outliers and thus possible issues. The primary purpose of this dashboard is the demonstration of the potential of DICOM metadata. For a concrete application or for a medical as well as technical analysis, specific dashboards can subsequently be constructed. The click-based filtering applies to all visualizations shown, it is always possible to include or exclude the desired fields within the lists and graphs for the cohort by selecting the "plus" or "minus" presented. Each of these actions results in the creation of a corresponding filter, which is displayed as a box at the top and can also be adjusted or disabled. It is also possible to create more complex search queries by manually adding specific filters.

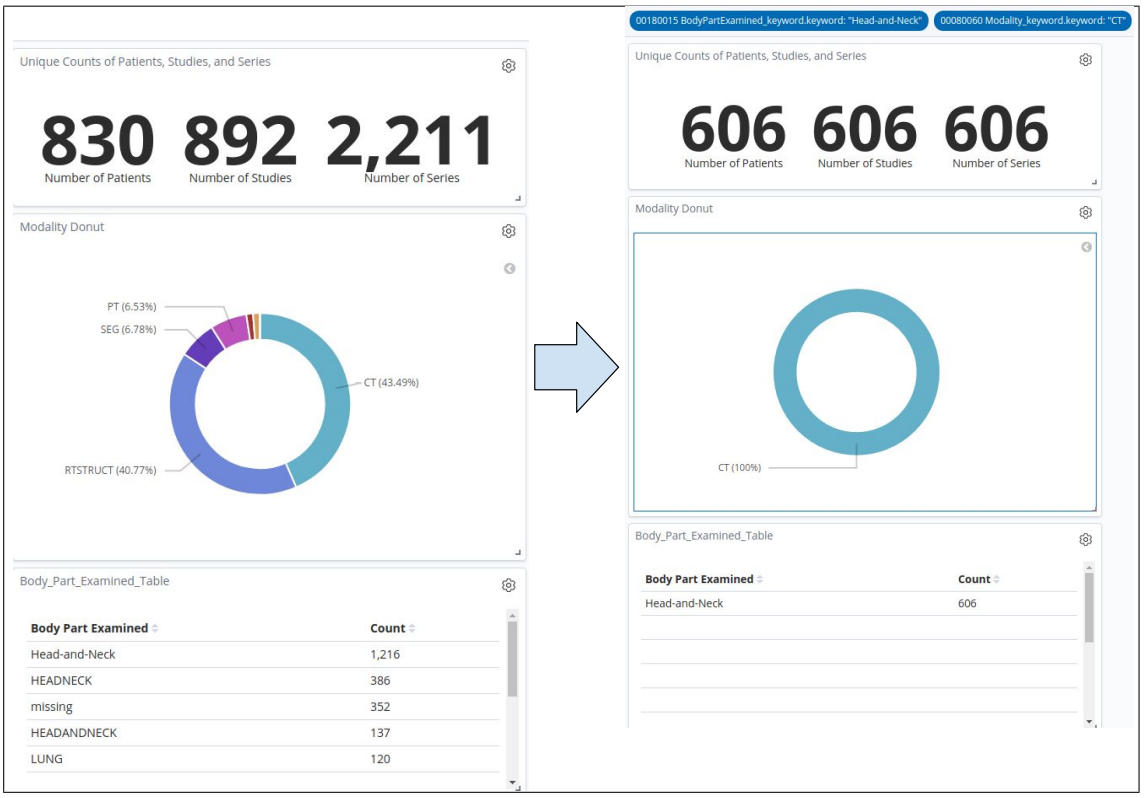

Figure 4.2: *The filtering of DICOM metadata in META.*

Figure 4.2 shows how the cohort for an algorithm, which uses CT images of the head and neck area as input data, can be defined. Here, by applying a filter on both the modality "CT" and the body part examined "Head and Neck", the data is reduced from initially over 2200 to 606 suitable images. Once all filters have been set and consequently the cohort defined, the desired analysis pipeline can be started.

This is done via the drop-down menu located at the top of the dashboard. Here, each entry of the list represents an installed pipeline, which are automatically fetched, so they always represent the current state of the platform. After the selection is done, a click on "start" opens the corresponding input dialog, which can be freely defined by the pipeline developers. Figure 4.32 shows an example of such a popup dialog, which, besides referencing the relevant publication, also provides information about the required input data or application areas. It is also possible to set the parameters, which should be considered during the processing. In this case, the model and architecture to be applied for the prediction can be chosen. Since these forms are always dynamically generated based on JSON schema, developers can define the layout themselves, and freely define check-boxes, text fields or drop-down menus as needed. Finally, the cohort can be limited to a specified amount of series, in case it is not necessary to handle all possible sequences. By pressing the button, all entries are validated and it is ensured that all required information has been entered. If this is the case, the form and the cohort query are sent to the FLOW section along with a status message, which is confirming the triggering of the pipeline.

The default configuration includes two additional META dashboards. The "Segmentation Dashboard" presents primarily information related to annotations. For this purpose, a filter is preset to the DICOM modality SEG, so that only series containing segmentations are displayed. Since they are often used as ground truth for the training of new models, this view is intended to provide detailed insights into the available annotations. Of particular interest in this context is how the segmentations were generated and which annotations are included in each case. Several visualizations list the underlying generator algorithms and also list each label that can be filtered individually. The number of available samples is crucial, since training is only reasonable above a certain amount of available samples.

The third "Trained Models Dashboard" has a focus on machine learning models encapsulated in DICOM. Besides the preselection of the data, it only lists some information about the creation of the model, such as the algorithm or the site ID of the creator. This view is especially valuable for installing and uninstalling models and for transferring them to other Kaapana-compatible instances. All dashboards also provide the ability to invoke "service pipelines" which allow, for example, deletion or downloading of the selected data.

4.2 Evaluation Environments

For the evaluation of a distributed infrastructure, it is essential to apply it in real scenarios. For this purpose, a platform instance was put into operation at each of the partner sites of two large German research consortia. Since these alliances are long-term efforts and are still evolving at the time of writing, they were mainly used to evaluate the establishment of the infrastructure as well as the conception of concrete projects. A complete federated analysis could not yet be carried out across the consortia, as the relevant local datasets were not yet ready. However, the necessary analysis pipelines for the specific project scenarios have already been developed and evaluated on public datasets using local servers at the DKFZ. The consortia and their background, members, missions and objectives are briefly described below.

4.2.1 The German Cancer Consortium (DKTK)

First, there is DKTK, which, as the initiator and funder of the initial JIP project, has a very important role in the overall development of this work. Since 2012, this initiative has been established by the German government to promote medical cancer research in Germany and acts as one of the six "German Centers of Health Research" with the participation of the Federal Ministry of Education and Research, the German federal states and the DKFZ. The consortium combines a total of more than 20 research institutions and university hospitals at seven locations. As the core center, the DKFZ serves as a coordinator between the sites and projects. DKTK aims to discover, develop, test and apply new personalized oncology strategies, with a focus on the fast translation of new, promising research outcomes into clinical application. The focus here is on promoting expertise in clinically oriented cancer research by building new long-term research infrastructures at the sites and attracting new talent by funding new professorships. Furthermore, multicenter collaboration and the implementation of clinical trials and projects, which have great potential due to the large number of scientists involved in the consortium, is a central mission. The DKTK's "Joint Funding Program" also enables the funding of such distributed projects like the JIP, of which more than 20 have already been launched since 2012 [DKTK, 2021].

The Joint Imaging Platform (JIP)

As already explained, medical imaging is a key part of cancer diagnosis, monitoring, and treatment, and that is why the JIP strategic initiative was launched in 2017.

The mission of the project is to improve the collaborative utilization of medical imaging in cancer research in Germany by establishing a platform where experts of the field can meet and pursue new joint endeavours. Figure 4.3 illustrates the JIP partner sites and how they are distributed across Germany. The 11 cooperation partners include the university hospitals of Dresden, Düsseldorf, Essen, Frankfurt, Freiburg, Heidelberg, Mainz, Tübingen, LMU Munich, TU Munich and the Charité Berlin. Next to the networking of experts and promotion of joint projects in the field of medical image computing, the JIP also aims to push forward the technical implementation of such a decentralized infrastructure that enables cutting-edge medical imaging research within the consortium, which was the starting point for this work. The project should enable standardized image analysis for clinical studies by allowing image-based patient stratification, therapy monitoring as well as radiomics analysis. Likewise, early detection and progression assessment will be accelerated and simplified by the consistent application and evaluation of the algorithms within the hospitals. Increasing incentives for collaboration through equal partners, maintaining data sovereignty, and sharing expertise are also important aspects in this context. Other IT platforms, which were already established within the DKTK as part of the CCP-IT [Lablans et al., 2015] should be connected to the JIP, so that both imaging information can be provided and information from other clinical systems can be retrieved.

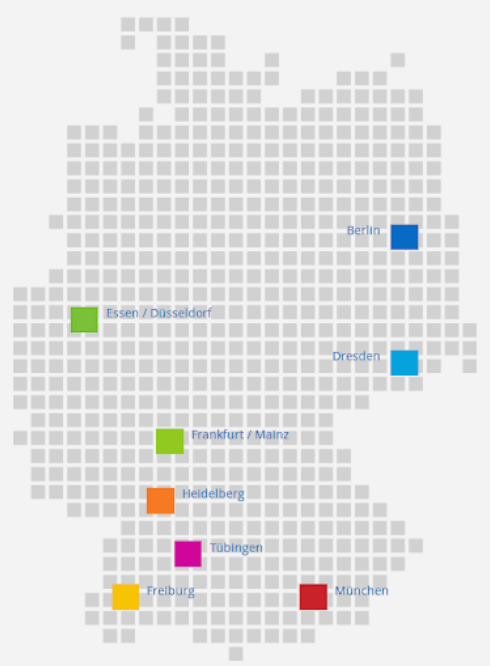

Figure 4.3: *A map of Germany indicating all DKTK partner sites.*

In the corresponding publication the JIP is described as follows:" The JIP is designed to facilitate collaborative imaging projects across institutions by addressing the typical technical, organizational, and legal challenges associated with the sharing of imaging data, acquisition parameters, analysis algorithms, or processing results. By enabling training, evaluation, and application of algorithms in large-scale federated clinical

settings, the platform builds a solid and extensible foundation for federated learning scenarios. Leveraging open-source technologies, the JIP has the potential to serve as a promoter of prospective cross-center radiologic studies at unprecedented cohort sizes, not only within the DKTK but also beyond." [Scherer et al., 2020b] The "beyond" in this description proved to be true, as it has become apparent in the meantime that many other projects could benefit from a similar infrastructure, which ultimately led to the Kaapana project. For this purpose, the functionality of the original JIP platform has been transformed into individually configurable components, which have been combined in a framework to be used for the configuration of project-specific platforms with customized features. Thus, most of the concepts and implementations described in this thesis originated in the JIP, which was taken as the basis for Kaapana.

Technical Infrastructure

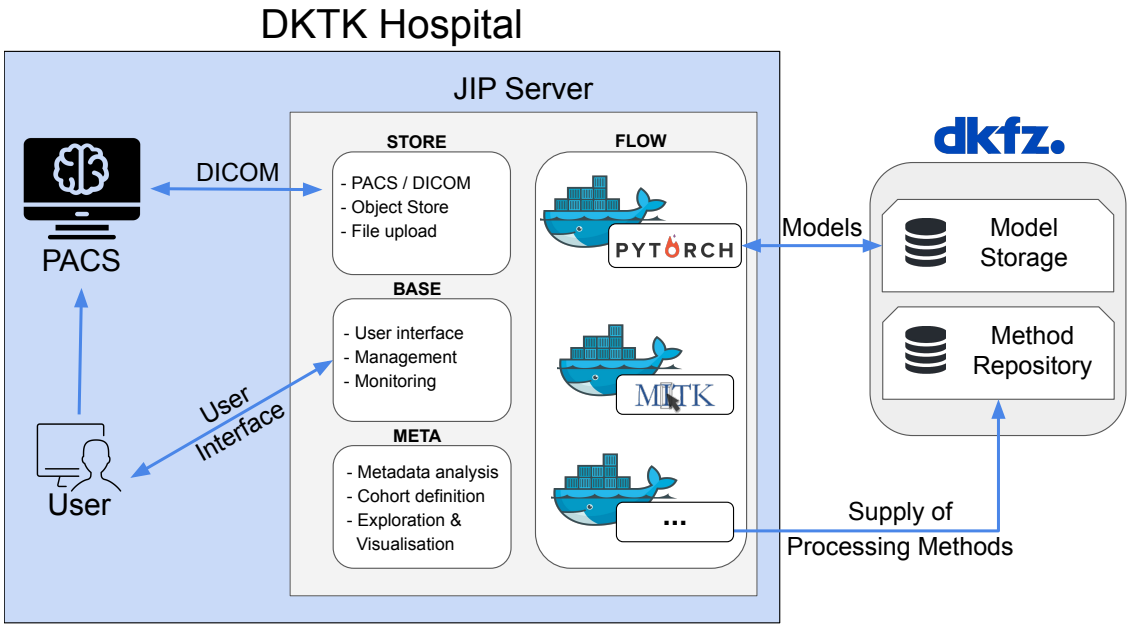

Figure 4.4: *Schematic representation of the JIP server within the clinical IT landscape.*

Figure 4.4 shows how the JIP is integrated at the hospitals of DKTK. The figure includes both the decentralized components of the JIP server at the hospital and the central hub operated at the DKFZ. The platform can only be accessed via the hospital's local network. Users interact with the UI via a web browser or send images to the system by transferring them from their local clinical PACS. Once the images arrive, they are located in the consistent environment, which provides all the features for standardized

processing. A central hub is provided to centrally provide platform updates, new processing methods or tools, which is operated as a Harbor [Harbor Authors, 2021] based container registry at DKFZ. Besides the provisioning of new features, it also serves as a central exchange channel for federated scenarios. However, the connection to the hub will only be opened for maintenance and installation of the platform software.

Server Hardware

CPU:	2x Intel Xeon Processor
RAM:	128 GB
GPU	2x NVIDIA TITAN Xp 12 GB GDDR5X
Storage:	2TB SSD + 10TB HDD in a RAID 1 configuration

Table 4.1: *Hardware specifications of the JIP servers for the hospitals.*

Besides the shared software, a uniform hardware is also important, as this ensures a standardized environment with comparable resources and runtimes. With a few exceptions, identical server configurations were chosen, which were delivered to the clinics. On the machines, CentOS was used as the OS, on which the JIP was installed. The servers have the specifications shown in table 4.1.

Project Milestones

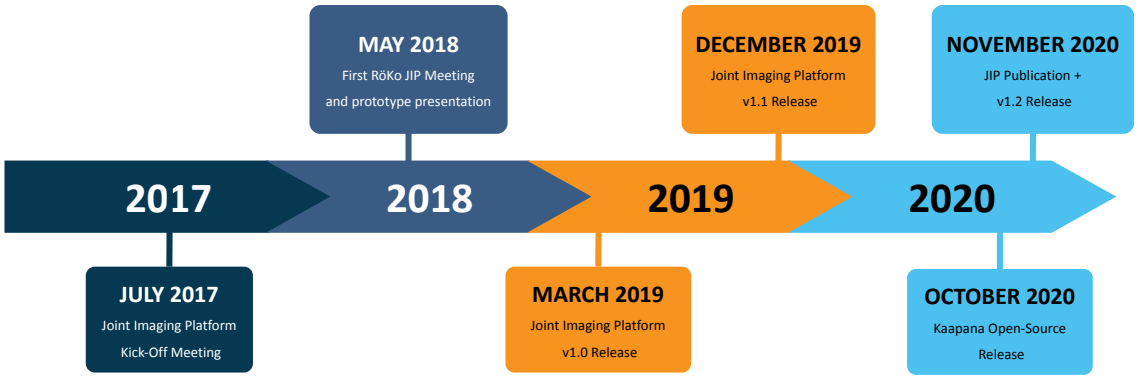

Figure 4.5: *Schematic timeline of the JIP project.*

The project started in July 2017 with a kick-off meeting in Heidelberg. In May of the following year, the first JIP meeting of the involved sites was held during the

Röntgenkongress (RöKo) in Leipzig, where also the first prototype of the platform software was presented. The meetings at the RöKo were continued in 2018 and 2019.

In March 2019, the first version of the JIP software was released, which could be put into operation by ten of the eleven sites. One site initially had organizational difficulties with the installation, which have since been resolved. The first update of the platform in December of the same year was also successfully completed by most sites and provided the first analysis methods with an automatic organ segmentation for abdominal CTs and a pipeline for Radiomics feature extraction. In 2020, the concepts of the prototype were refined and the JIP was transferred into the framework Kaapana, which was also published as an open source project together with a new JIP version. Furthermore, the publication "Joint Imaging Platform for Federated Clinical Data Analytics" could be published at the JCO Clinical Cancer Informatics [Scherer et al., 2020b]. Within the DKTK, the JIP is already utilized to support some trials, which are briefly outlined below:

ARMANI: A prospective trial to compare anatomic v wedge resection of liver metastases in patients with RAS-mutated colorectal cancer.

MEMORI: (ClinicalTrials.gov identifier: NCT02287129) Correlation of pre- and post-treatment PET/CT imaging biomarkers with histopathologic and molecular features in esophagogastric adenocarcinomas.

NEOLAP: (German Clinical Trials Register identifier: DRKS00019011) Development and application of machine learning algorithms for noninvasive image-based subtyping of locally advanced pancreatic adenocarcinoma.

PROACTIVE: (German Clinical Trials Register identifier: DRKS00013915) A prospective longitudinal examination of patients with low-risk prostate cancer aiming at investigating the benefits of multiparametric MRI for active surveillance.

qPSMA-JIP: Retrospective evaluation of the prognostic value of PSMA-targeted PET/CT for clinical outcome in patients with metastatic hormone-sensitive and castration-resistant prostate cancer.

DKTK Radiation Oncology Group (DKTK ROG): Explorative study to predict locoregional control after postoperative chemoradiotherapy of locally advanced oropharyngeal carcinoma based on HPV-16 DNA status.

4.2.2 The Radiological Cooperative Network (RACOON)

In response to the COVID-19 pandemic, in 2020 the Network of University Medicine (NUM), a network of all university hospitals of Germany, has been established. The mission of NUM is to bundle and strengthen the activities of all 36 German university hospitals in order to improve the handling and control of the pandemic. It is funded by the German Federal Ministry of Education and Research (BMBF) and received 150 million euros funding for the first year. A total of 13 projects were initially supported, all of which investigate different aspects of pandemic response. Since the majority of severe SARS-CoV-2 infections have pulmonary involvement, and the disease progress can be well differentiated and monitored by radiological findings, the Radiological Cooperative Network (RACOON) was launched within NUM to specifically address the radiological aspects of the pandemic. For a quantitative and systematic evaluation of the radiological data using automated methods, however, a standardized acquisition of the findings must be achieved. For several years, so-called structured reporting has been established, linking each finding and measurement with meta-information that precisely and reproduceably defines how a finding was collected, quantified, and derived from other data.

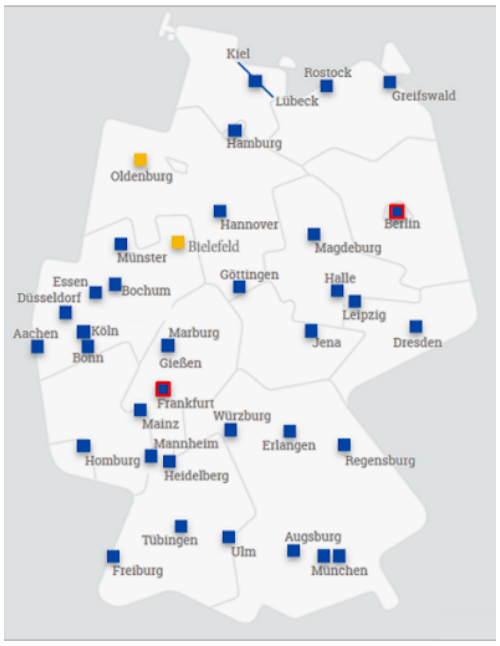

Figure 4.6: *A map of Germany indicating all RACOON partner sites.*

The RACOON network aims to establish such structured reporting by establishing a Germany-wide infrastructure for the standardized collection of radiological data, and to utilize this for pandemic response [NUM, 2021]. This infrastructure will not only enable structured and uniform reporting, but also enable automatic data processing, which should generate new insights through the application and training of state of the art analytical methods. The project is led by the university hospitals Charité Berlin and Frankfurt, which together with the DKFZ, the Technical University of Darmstadt (TUDa), the Fraunhofer Institute for Digital Medicine (MEVIS) and the Mint Medical GmbH build the infrastructure needed. Figure 4.6 illustrates the geographical locations of all 36 participating university hospitals distributed across Germany.

Technical Infrastructure

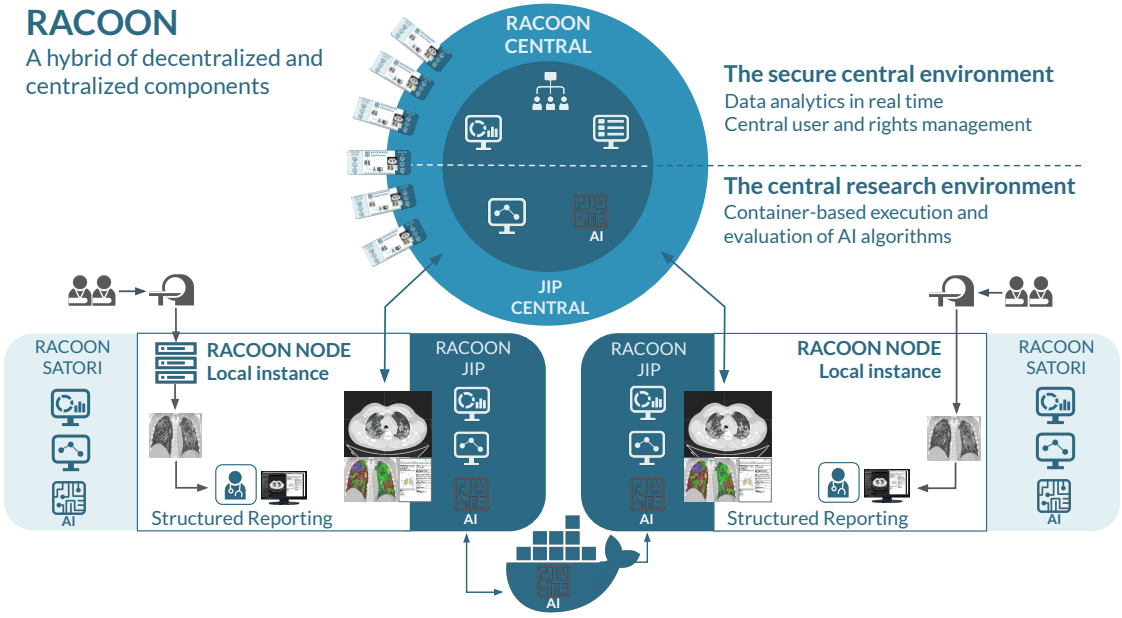

Figure 4.7: *Schematic representation of the RACOON infrastructure.*

Figure 4.7 illustrates the hybrid technical infrastructure as it is deployed for RACOON. It is hybrid because the architecture provides central components as well as decentralized instances at the respective sites. For both aspects, different tasks are anticipated, which are presented as follows.

RACOON NODE
The RACOON NODEs form the backbone of the decentralized infrastructure where each participating hospital operates a dedicated server as individual instance, where each of them provides the three sections: RACOON REPORTING, RACOON JIP and RACOON SATORI.

RACOON REPORTING: Serves as a local assesment station, which is used at every hospital to enable structured reporting for COVID-19 patients and to provide a consistent data basis. The software used is a certified medical product, which meets all requirements for clinical use. A custom RACOON reporting-template has been provided to ensure that all relevant information is captured. During local assessments, non-anonymized patient data is available, which can be enhanced by information from other data sources, such as the Radiological Information System (RIS) or the

Hospital Information System (HIS). Furthermore, the software allows the upload of data to the central infrastructure of RACOON CENTRAL. Here, it is important that this is only possible on explicit request and manual triggering of the corresponding staff of the respective clinic. Before each upload, the data is anonymized according to the "Attribute Confidentiality profile" [NEMA, 2021a] as defined by the DICOM standard. Additionally, the anonymized data can also be forwarded to the two other local instruments, SATORI and JIP. All of them are running on the same server and communicate via DICOM with REPORTING.

RACOON SATORI: Is a research tool developed and provided by MEVIS for the efficient annotation of three-dimensional images. It offers dedicated tools which facilitate manual segmentation or the revision of already existing annotations. Besides segmentation, it is also possible to automatically analyze the images for certain quality properties regarding signal intensity, homogeneity and image artifacts. Similarly to the other components, data is received via a DICOM receiver and results are also provided accordingly, which ensures interoperability as well as allowing communication between all the tools.

RACOON JIP: The platform created as part of the DKTK JIP project was also integrated as a component in all RACOON NODES. Similar to the initial project, it is used here for the exploration and analysis of imaging data within the consortium. Segmentation workflows and Radiomics feature extraction are of particular interest, as this could assist in the assessment of SARS-CoV-2 infections. Furthermore, the JIP also aims to enable federated learning between the RACOON partner sites.

To ensure a high level of data privacy, the JIP only receives anonymized data via RACOON REPORTING, which is feasible since many of the algorithms do not necessarily require personal metadata. This instance of JIP has no external communication - in fact, not even with RACOON CENTRAL and as such the only communication channel is within the NODE via REPORTING.

During the project two third-party analysis pipelines were developed by TUDa and the University Hospital Frankfurt which were integrated into the platform. Here, COVID-19-specific tissue alterations are detected, segmented, and associated volumes are determined. The analysis results are then kept as DICOM SEG and DICOM SR within the internal storage. The second pipeline aggregates the COVID-19 specific image features across the site cohort and transfers them to RACOON CENTRAL via DICOM SR, where they are finally visualized in a dashboard provided by RACOON JIP CENTRAL. More integration and implementation details follow in the section of the scenario evaluation.

RACOON CENTRAL
In this infrastructure there is also a cloud-based central component called RACOON CENTRAL, which is primarily used to enable data exchange and central data collection for the association. Two scenarios are particularly relevant here: First, anonymized images from the local sites can be uploaded to the cloud instance for joint evaluations as part of multicenter studies. The second use case allows re-evaluation and consultation between medical professionals for specific cases. As described, the RACOON NODEs are responsible for the transfer of anonymized data between the decentralized and centralized subsystems. In contrast to the decentralized instances, the central instance is designed for cross-site workflows and clinical trial reporting.

RACOON JIP CENTRAL
Similar to the local instances, next to the reporting and data administration capabilities, also RACOON JIP CENTRAL is deployed as a central processing instance. It's mainly responsible for cross-site data analysis, which processes the anonymized data uploaded by the sites. Like within the local instances, data is transferred between the JIP and RACOON CENTRAL via DICOM.

Since all sites can contribute images, a valuable broad database can be established by providing real datasets representing all of Germany. In addition to the large amount of high quality images that can be collected, the heterogeneity of the data is also significantly increased by the numerous acquisition locations, machines, and settings. These centrally executed analyses should allow for a significantly more realistic evaluation of method performance, as they are based on data from real hospitals in real world scenarios. Also, for the training of new models the broad database offers advantages, as it should allow to generate more robust predictions, which perform well on data from most hospitals. For the execution of federated use cases, JIP CENTRAL represents the intermediate layer, which handles the cross-linking of the different sites.

Server Hardware
Since instances of the RACOON infrastructure, unlike the DKTK JIP, consist of several components, a different deployment model has been chosen. For the local RACOON NODEs, three software components must be operated on each hospital server: Mint Medical GmbH's product for RACOON REPORTING, SATORI and the JIP. Since they have different system requirements in terms of operating system, software dependencies and network connectivity, a different approach was taken, providing each component with its own virtual machine (VM) on the server. Since all three components communicate via network only, they can be easily operated and maintained in their own isolated environments. This also allows the UIs of the individual systems to be

configured separately via port forwarding on the server. The server hardware of most hospitals (some have made customizations) has the following specifications:

CPU:	AMD EPYC™ ROME 7702P, 64-Core
RAM:	256GB DDR4
GPU	NVIDIA Quadro RTX 6000 - 24 GB GDDR6
Storage:	14TB SSD

Table 4.2: *Hardware specifications of the RACOON NODE servers for the hospitals.*

Based on this, Windows Server 2019 was selected as OS, which offers HyperV as a VM solution and enables GPU passthrough. Thus, each server runs three separate VMs, allocating the components corresponding resources as disk space, CPU, RAM and GPU. For installation, the full JIP VM has been provided as a package, which was transferred and launched at the sites.

4.2.3 Local Development Infrastructure

For the local development of the JIP and evaluation of processing pipelines, also at DKFZ development servers have been established and populated with public datasets (see table 4.5). Even though such a local setup cannot provide realistic results for the medical evaluations due to the lack of the corresponding patient data, tests can still help to optimize and verify the functioning of the methods as well as the tuning to the hardware components.

The purpose of the local evaluation is less focused on the investigation of specific medical questions, but rather on verifying the effectiveness of the infrastructure and its concepts. The server hardware used for the local evaluations is similar to the hardware deployed within the consortia, which allows comparable workloads and runtimes of the tests and the real execution at the local sites.

DKFZ JIP server:
For the evaluation on the JIP server, CentOS 7 as OS was installed on a bare-metal server and the platform was set up with the default installation scripts.

CPU:	2x Intel Xeon Processor
RAM:	128 GB
GPU	2x NVIDIA TITAN Xp 12 GB GDDR5X
Storage:	2TB SSD + 10TB HDD in a RAID 1 configuration

Table 4.3: *Hardware specifications of the JIP development server at DKFZ*

DKFZ RACOON Server:
The test setup for RACOON also runs Windows Server 2019 as OS and provides RACOON REPORTING, SATORI and JIP within independent VMs. Despite the development server's hardware being better equipped than the RACOON NODEs, an identical setup can be reproduced by allocating the resources to the VMs.

CPU:	AMD EPYC™ ROME 7702P, 64-Core
RAM:	1TB DDR4
GPU	4x NVIDIA Quadro RTX 6000 a 24 GB GDDR6
Storage:	14TB NVME-SSD

Table 4.4: *Hardware specifications of the RACOON development server at DKFZ*

Datasets Utilized:
For the local tests during the development of the RACOON infrastructure, the test server was populated with approximately 25000 series from the TCIA, which are listed in table 4.5.

Full Title	**Data Citation**
Stony Brook University COVID-19 Positive Cases	[Saltz et al., 2021]
Breast-MRI-NACT-Pilot	[Newitt and Hylton, 2016]
The Cancer Genome Atlas Kidney Renal Clear Cell Carcinoma	[Akin et al., 2016]
The Cancer Genome Atlas Liver Hepatocellular Carcinoma	[Erickson et al., 2016]
head and neck squamous cell carcinoma	[Grossberg et al., 2020]
Quantitative Imaging Network head and neck cancer	[Beichel et al., 2015]

Table 4.5: *Table of datasets used for the development of the RACOON infrastructure*

4.3 Use Case Evaluations

For the evaluation of the use cases and their scenarios (see 3.1.5), real-world examples were collected and implemented. These integrations have been implemented by both internal Kaapana developers as well as external partners, which is important as the requirements imply usability by non-specialists from within the community.

4.3.1 Use Case 1: Autonomous Execution

This use case addresses the enabling of autonomous processing decisions, which, as described in the main concept of the methods chapter, should allow autonomous decision making by the platform to trigger pipelines in case defined conditions are met. For each incoming DICOM-series it is checked if trigger automatic processing rules are present. If so, the rules are applied to the image to initiate processing if necessary.

Case Scenario 1.1: Existing Metadata

For this case scenario, the RACOON task from case scenario 1.4 is also used, since it deals with the analysis of incoming CTs, which should be triggered automatically if a corresponding image arrives at the platform. The associated DICOM-tag "Modality" represents a standardized and thus very reliable entry which is usually set automatically by the scanner. As a consequence, a rather basic auto-trigger configuration is sufficient in this case - *dag_racoon_ukf_presegmentation_trigger_rule.json:*

```
{"search_tags": {
        "0x0008,0x0060": "CT"
    },
    "dag_ids": {
        "racoon-ukf-presegmentation": {
            "rest_call": {
                "global": {
                        "single_execution": true
                }
            }
        }
    }
}
```

The important element here is the requirement of the DICOM Header (0008,0060) Modality corresponding to the entry "CT". Since within RACOON all data is forwarded via RACOON REPORTING, and thus the pre-selection of relevant images is already made there, the scenario is already fulfilled with this configuration. However, it is also evident that more complex requirements could also be realized by adding additional DICOM header entries, which are then all checked for compliance. The other settings indicate which DAG is to be triggered and it is also possible to define parameters for the processing pipline for automatic triggering.

Case Scenario 1.2: Pre-analysis and Metadata Extraction

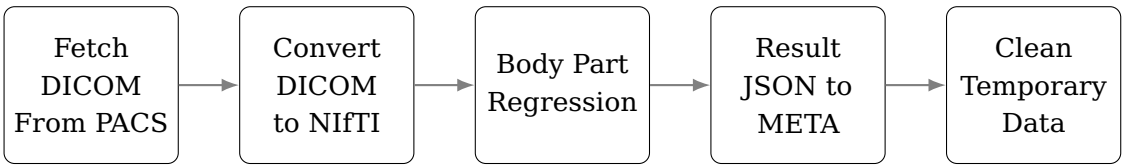

Figure 4.8: *Processing pipeline for the body part prediction.*

In contrast to the first scenario, an autonomous trigger decision is much more challenging when the metadata is incomplete, non-standardized or hospital-specific. As described in the use case "Autonomous Execution" of the methods chapter (see 3.1.5), the modality and the body part examined are the crucial input data parameters for most algorithms. Since the modality generally is reliable DICOM information, the body part examined was of greatest interest for the evaluation of the second scenario. For this purpose, a body part regression model for CT volumes has been developed, which was trained in a self-supervised fashion, and predicts the desired information as a pre-processing step, instead of relying on the information provided. Figure 4.8 illustrates the related processing pipeline, which after conversion from DICOM to NIfTI is predicting the corresponding body region. In the last step, the prediction is added to the META system so that this information is also available for cohort selection within the dashboards as shown in figure 4.9. This prediction results in defined body sections instead of the usual non standardized entries, which allows precise definition from where to where the body was captured on each series. For this purpose, the regions HEAD,NECK,CHEST,ABDOMEN and PELVIS were chosen, which are added to the body part predicted tag according to their occurrence.

Body Part Predicted

Body Part Predicted	Count
CHEST	1,949
ABDOMEN	1,543
HEAD-NECK	1,453
ABDOMEN-PELVIS	998
HEAD-NECK-CHEST-ABDOMEN-PELVIS	555
CHEST-ABDOMEN-PELVIS	497
NONE	495
NECK-CHEST-ABDOMEN-PELVIS	234
NECK-CHEST	198
PELVIS	183

Figure 4.9: *Dashboard visualization for the predicted body parts.*

If the lung analysis from scenario 1 should be performed on non-RACOON data, the applicable images can be selected by filtering for modality CT and a Body Part Predicted including the CHEST. For the data on the test system, this translates to instead of the 2106 series that can be identified by the included DICOM tags Modaily CT and Body Part Examined equals CHEST or LUNG, the standardized prediction identifies 2936 matching series including the CHEST area. It is also likely that the selection will be

more accurate as well, since all images are reviewed equally. Examples in the datasets, on the other hand, frequently show that despite identical descriptions of the body part or series descriptions, the series can still contain varying scans. Due to the lack of ground truth, a quantitative assessment of this accuracy is not possible and can only be verified by random sampling.

4.3.2 Use Case 2: Integration of Data Analysis Algorithms

This use case addresses the integration of new processing steps, and is therefore of great importance for the accomplishment of Kaapana's main goals. It is particularly important that the concepts developed are on the one hand simple and straightforward enough to be implemented by non-Kaapana experts, but on the other hand still offer enough flexibility and freedom for the integration of complex processes that have not yet been considered during the development of the framework. For the evaluation of this use case, three processing pipelines developed internally at DKFZ representing different styles of data processing were examined, as well as a third-party pipeline to evaluate off-site applicability.

Case Scenario 2.1: CPU-based ML Model Inference

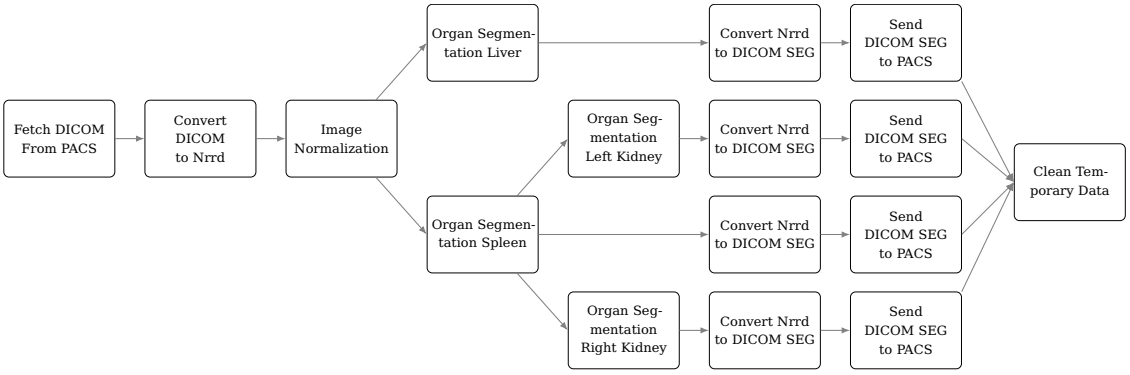

Figure 4.10: *Processing pipeline for the 3D-SSM based organ segmentation.*

For the initial integration of an analysis method, a 3D Statistical Shape Models (3D-SSM) [Norajitra and Maier-Hein, 2017] based organ segmentation has been chosen, which generates segmentation masks for the liver and spleen, as well as the left and right kidney from abdominal CT imaging. Figure 4.10 shows the complex structure of this processing pipeline, consisting of many individual prediction steps. The key characteristic of this workflow is that everything can be executed on the CPU only, which means that, unlike most newer DL-based approaches, hardware without a GPU is also usable.

Input Data Selection

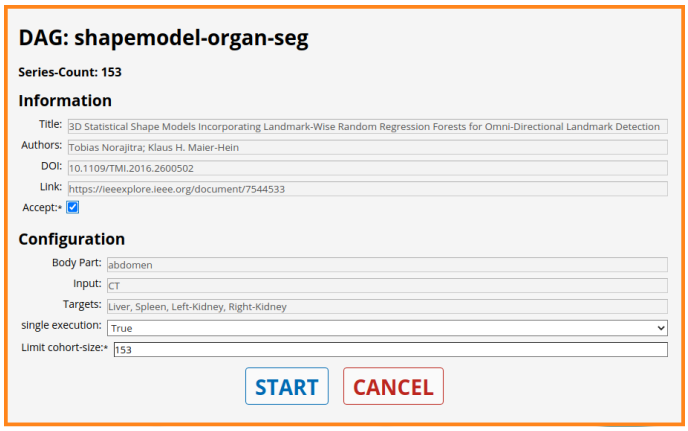

Figure 4.11: *Trigger dialog for the 3D-SSM organ segmentation DAG.*

The pre-trained model of this method expects abdominal CT images, which are available in the Nrrd file format. Thus, data selection can be easily achieved using two filters with modality equals CT and Body Part Examined equals abdomen on the META dashboards. The triggering dialog shown in Figure 4.11 includes information about the corresponding publication, the expected input data and the target segmentations of liver, spleen, left and right kidney.

Processing Pipeline

After the provisioning of DICOM images from the PACS, all images are converted by the *DcmConverterOperator* into the Nrrd format. The original implementation of the algorithm first normalized all images, then sequentially segmented one organ at a time, which has negative consequences for the execution time. Since Kaapana allows parallel execution of Operators, this procedure was modified during the integration-process and changed into five individual processing steps, partially parallelized where possible. After normalization, parallel inference with the liver and spleen models is performed. The prediction of the left and right kidney cannot be done at the same time, as it depends on the resulting liver and spleen positions, and thus have to wait for the outcome of these predictions. However, once the initial predictions are completed, the kidneys are also segmented in parallel.

The models produce four Nrrd files each containing the respective organ mask. The files are subsequently converted to DICOM SEG using the *Itk2DcmSegOperator* and sent to the internal PACS as the final result using the default *DcmSendOperator*. On arrival, similar to other incoming images, the metadata is extracted and presented in the META dashboards for data exploration. All processing steps, related to the actual method to be integrated, were realized using a single container, performing the individual tasks by specifying distinctive arguments. It is based on MITK, which has been used to implement the respective normalization and organ predictions. All other tasks were realized by default Operators from the Kaapana software catalog.

For CPU-based processing jobs, RAM is usually the resource bottleneck since image processing is usually very memory-intensive and CPU cores can be better distributed among different workloads. In this case, a consumption of 6 GB RAM per prediction is specified for the scheduling system, which schedules the execution depending on the current hardware load.

Results

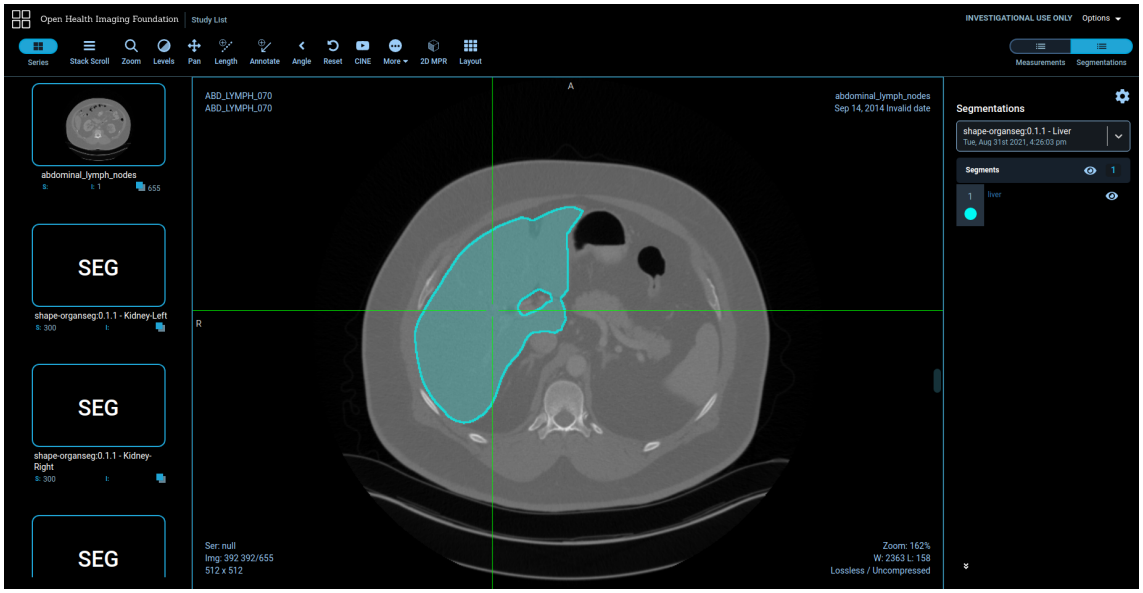

Figure 4.12: *OHIF showing the segmentations produced by the 3D-SSM organ segmentation.*

As shown in figure 4.12, the outcome of the image analysis can be viewed in OHIF, where each organ is shown as a single-label DICOM SEG, and its mask presented as an overlay on top of the CT scan. Furthermore, they also appear on META's specialized segmentation dashboard, where they can be selected for follow-up analysis.

Case Scenario 2.2: GPU-based ML Model Inference

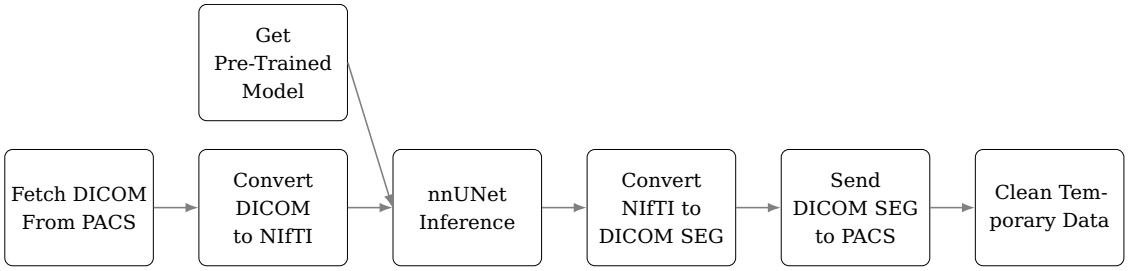

Figure 4.13: *Processing pipeline for the nnUNet inference.*

The second case scenario addresses the integration of DL-based algorithms requiring intensive GPU allocation due to the underlying technology. Given the successful participation in many scientific challenges, the versatile applicability, and the availability of several pre-trained models, the nnUnet inference [Isensee et al., 2021] has been selected for integration for this scenario. The processing steps shown in Figure 4.13 illustrate the implementation of the nnUNet-prediction DAG in Kaapana. In contrast to scenario one, it is much more straightforward in design, since all segmentations can be generated during a single model inference.

Input Data Selection

The selection of the input data is handled via META, where depending on the selected task, varying input data is required, which is shown in the corresponding dialog.

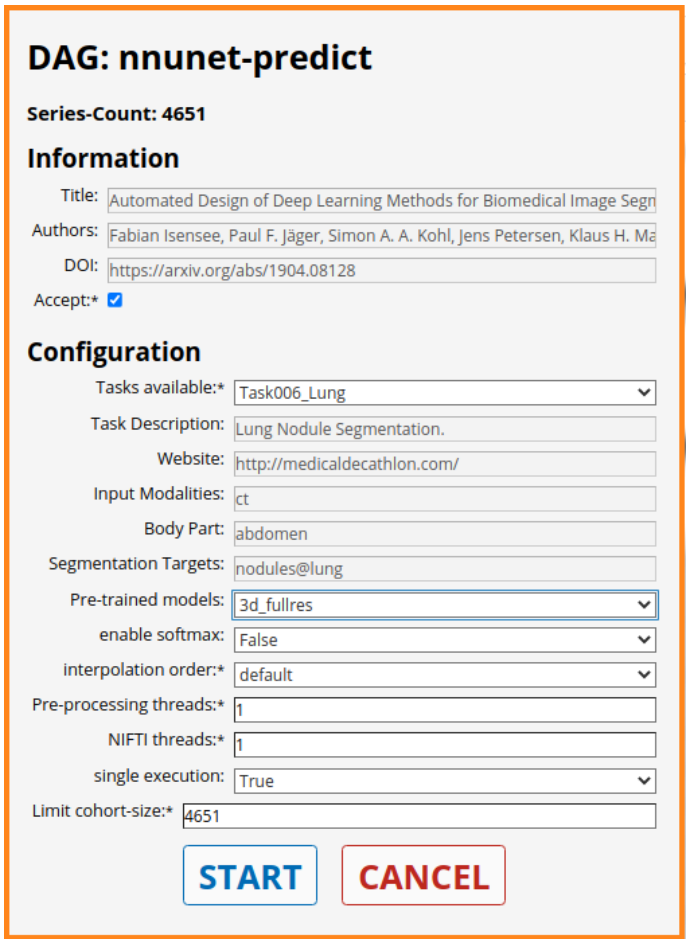

Figure 4.14: *Trigger dialog of the nnUNet DAG.*

However, this time there are no general image properties that apply for the DAG overall - each supported task has its own requirements. The information, which data is required for the respective model (or task), is also provided via the triggering dialog, where the corresponding entries are adjusted according to the selected task. Figure 4.14 shows the layout of the dialog with all the information and the adjustable parameters, such as the task selection. Here, tasks and models are distinguished, as the task specifies the function (e.g., input data and target structures), whereas the model explicitly sets the architecture for the model of the inference. Most pretrained tasks provide models for 2d, 3d-lowres, and 3d-fullres architectures. The model list includes models installed on the server as well as those available for download and supported by the current implementation. After the launch, along with the usual preparation of relevant DICOM series, the availability of the selected model is checked in parallel and if unavailable, the download is automatically triggered.

Processing Pipeline

The implementation of the nnUNet Operator has been realized using a single container, which not only supports inference with various models, but also data pre-processing and the training of new models, which is described in use case 5.

The container is based on a special base-image provided by NVIDIA [NVIDIA NGC, 2021], which allows GPU access and also provides Pytorch [Paszke et al., 2019] as a DL framework. By specifying the required GPU resources to the nnUNet Operator, the scheduling system manages the execution of the containers to ensure sufficient GPU RAM is always available for the job. In this case, it is expected that this is the bottleneck for the execution of the inference on the system, since the GPU memory is usually very limited. However, the other checks of the scheduler prevent the system from being overloaded by the other demands as well. The resulting NIfTI files may contain multiple segmentations and are therefore subsequently converted into multi-label DICOM SEG using the *Itk2DcmSegOperator* and finally sent with the *DcmSendOperator* to the internal PACS.

Results

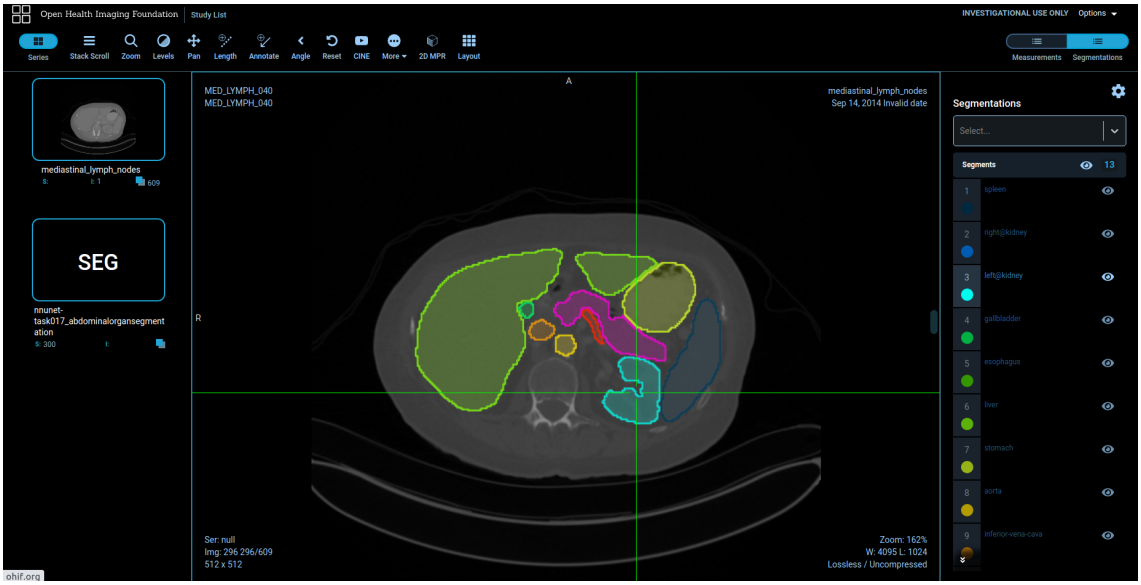

Figure 4.15: *OHIF showing the multi-label segmentations produced by the the nnUNet inference DAG using the task17 pre-trained model.*

Screenshot 4.15 illustrates OHIF showing the segmentations generated by the nnUNet "Task017_AbdominalOrganSegmentation" model, which segments the **spleen,left and right kidney, gallbladder, esophagus, liver, stomach, aorta, inferior-vena-cava, portal-vein, splenic-vein, pancreas and the left and right adrenal-gland** within abdominal CT Series. Since multi-label segmentations are used, only one DICOM SEG is stored in the PACS and presented here.

Case Scenario 2.3: Radiomics Feature Extraction

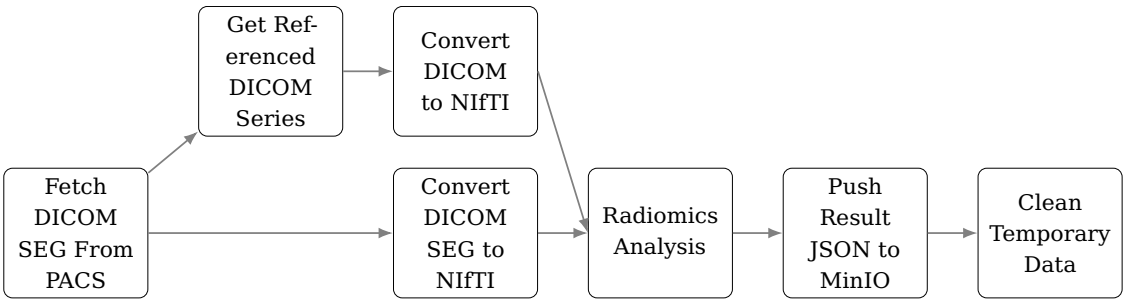

Figure 4.16: *Processing pipeline for the Radiomics feature extraction.*

For the evaluation of the integration of non-ML based approaches, a Radiomics feature extraction pipeline has been implemented in this case scenario. Besides the different type of analysis, this DAG also differs in the output format, as no voxel-based result is generated here, but rather textual statistical features.

Input Data Selection

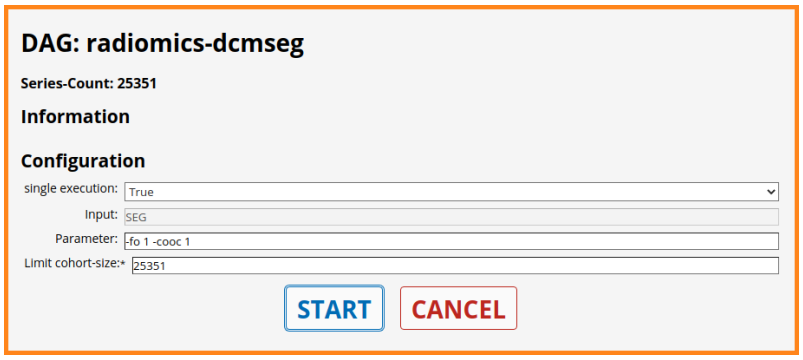

Figure 4.17: *Trigger dialog for the Radiomics DAG.*

The selection of input data is facilitated by the specialized META segmentation dashboard, which exclusively presents relevant data and its detailed properties. The associated trigger dialog shown in Figure 4.17 also allows the specification of parameters for the execution of the Radiomics application, which specify what features to be calculated.

Processing Pipeline

The DAG, as shown in Figure 4.16, first retrieves the selected DICOM SEG objects from the PACS and converts them into Nrrd files using the *DcmSeg2ItkOperator*. In parallel, the *LocalGetRefSeriesOperator* is used to identify the corresponding CT or MRI images where the segmentations were generated from. Once these have also been downloaded and converted to NIfTI, both the masks and the associated images are fed into the Radiomics Operator. In this case, the results of two processing steps are supplied to an Operator, which can then evaluate them together. The Radiomics Operator is based on a container including MITK Phenotyping [Götz et al., 2019] and its *MitkCLGlobalImageFeatures* application, which uses the segmentation mask to calculate the requested features on the image and provides them as a XML file.

Results

Since there is currently no practical solution in Kaapana for the storage of these characteristics as DICOM, the results are not saved in the PACS as usual, but instead stored directly as a XML file in the integrated Object Storage MinIO using the *LocalMinioOperator*, where they can be downloaded and analyzed onwards via the REST API or UI referenced by the corresponding SeriesUID of the DICOM SEG. Figure 4.18 shows parts of the XML representation containing the first order statistic obtained from a Radiomics analysis using a liver segmentation mask.

```
</mp:parameters>
▼<mp:features>
  <mp:feature name="Mean" version="1" class="First Order" setting="fo">69.5261</mp:feature>
  <mp:feature name="Unbiased Variance" version="1" class="First Order" setting="fo">501.492</mp:feature>
  <mp:feature name="Biased Variance" version="1" class="First Order" setting="fo">525.673</mp:feature>
  <mp:feature name="Skewness" version="1" class="First Order" setting="fo">-1.26255</mp:feature>
  <mp:feature name="Kurtosis" version="1" class="First Order" setting="fo">18.8561</mp:feature>
  <mp:feature name="Median" version="1" class="First Order" setting="fo">77.5625</mp:feature>
  <mp:feature name="Minimum" version="1" class="First Order" setting="fo">-854</mp:feature>
  <mp:feature name="Maximum" version="1" class="First Order" setting="fo">192</mp:feature>
  <mp:feature name="Range" version="1" class="First Order" setting="fo">1046</mp:feature>
  <mp:feature name="Mean Absolute Deviation" version="1" class="First Order" setting="fo">17.4681</mp:feature>
  <mp:feature name="Robust Mean Absolute Deviation" version="1" class="First Order" setting="fo">13.1242</mp:feature>
  <mp:feature name="Median Absolute Deviation" version="1" class="First Order" setting="fo">17.1964</mp:feature>
  <mp:feature name="Coefficient Of Variation" version="1" class="First Order" setting="fo">0.329769</mp:feature>
  <mp:feature name="Quantile Coefficient Of Dispersion" version="1" class="First Order" setting="fo">0.194612</mp:feature>
  <mp:feature name="Energy" version="1" class="First Order" setting="fo">1.9596e+10</mp:feature>
  <mp:feature name="Root Mean Square" version="1" class="First Order" setting="fo">73.2359</mp:feature>
  <mp:feature name="Robust Mean" version="1" class="First Order" setting="fo">70.1683</mp:feature>
  <mp:feature name="Uniformity" version="1" class="First Order" setting="fo">0.213806</mp:feature>
  <mp:feature name="Entropy" version="1" class="First Order" setting="fo">2.51999</mp:feature>
  <mp:feature name="Excess Kurtosis" version="1" class="First Order" setting="fo">15.8561</mp:feature>
  <mp:feature name="Covered Image Intensity Range" version="1" class="First Order" setting="fo">0.2615</mp:feature>
  <mp:feature name="Sum" version="1" class="First Order" setting="fo">2.5402e+08</mp:feature>
  <mp:feature name="Mode" version="1" class="First Order" setting="fo">77.5625</mp:feature>
  <mp:feature name="Mode Probability" version="1" class="First Order" setting="fo">1.1022e+06</mp:feature>
  <mp:feature name="Unbiased Standard deviation" version="1" class="First Order" setting="fo">22.394</mp:feature>
  <mp:feature name="Biased Standard deviation" version="1" class="First Order" setting="fo">22.9276</mp:feature>
  <mp:feature name="Number Of Voxels" version="1" class="First Order" setting="fo">3.65359e+06</mp:feature>
  <mp:feature name="05th Percentile" version="1" class="First Order" setting="fo">32.9969</mp:feature>
  <mp:feature name="10th Percentile" version="1" class="First Order" setting="fo">42.1693</mp:feature>
  <mp:feature name="15th Percentile" version="1" class="First Order" setting="fo">47.5348</mp:feature>
  <mp:feature name="20th Percentile" version="1" class="First Order" setting="fo">52.9003</mp:feature>
  <mp:feature name="25th Percentile" version="1" class="First Order" setting="fo">56.3442</mp:feature>
  <mp:feature name="30th Percentile" version="1" class="First Order" setting="fo">59.2198</mp:feature>
  <mp:feature name="35th Percentile" version="1" class="First Order" setting="fo">62.0953</mp:feature>
  <mp:feature name="40th Percentile" version="1" class="First Order" setting="fo">64.9709</mp:feature>
  <mp:feature name="45th Percentile" version="1" class="First Order" setting="fo">67.8465</mp:feature>
  <mp:feature name="50th Percentile" version="1" class="First Order" setting="fo">70.6254</mp:feature>
  <mp:feature name="55th Percentile" version="1" class="First Order" setting="fo">73.2151</mp:feature>
  <mp:feature name="60th Percentile" version="1" class="First Order" setting="fo">75.8048</mp:feature>
```

Figure 4.18: *Sample set of the extracted Radiomics features from case scenario 1.3.*

Case Scenario 2.4: Third-Party Workflow Development

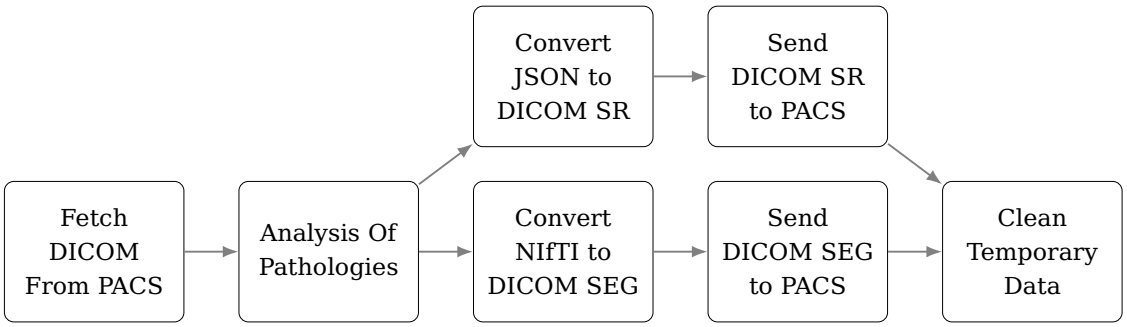

Figure 4.19: *Processing pipeline for the extraction of lung pathologies.*

For the evaluation of the integrability of methods by external community members, a task from the RACOON consortium has been selected. This analysis segments the typical lung pathologies associated with SARS-CoV-2 infections such as Ground-glass opacity (GGO), non-malignant lung consolidations, and pleural effusions from a patient's CT image and also extracts the respective volumes of the detected structures.

Input Data Selection
Even though this DAG can still be started via the dashboard as usual, for RACOON it should run automatically when suitable images arrive on the platform, as it is described in the first use case. In general the DAG expects CT scans of the chest area as input images.

Processing Pipeline
The figure 4.19 illustrates the workflow design, which was developed in collaboration between the TU Darmstadt and the University Hospital Frankfurt. The prediction is enabled by a container developed by the RACOON partners, which directly processes the DICOM files and produces both a NIfTI file containing the segmentations and a JSON file including the extracted volumes of extracted tissues. The segmentation is converted into a multi-label DICOM SEG using the *Itk2DcmSegOperator* and sent to the internal PACS using the D*cmSendOperator*. The JSON gets converted to DICOM SR using the *Json2DcmSROperator* and is also finally sent to the internal PACS.

Results
As usual, the DICOM results can be inspected via OHIF. Figure 4.20 illustrates that aside from the CT with the segmentation overlays, also the DICOM SR documents can be visualized in the browser.

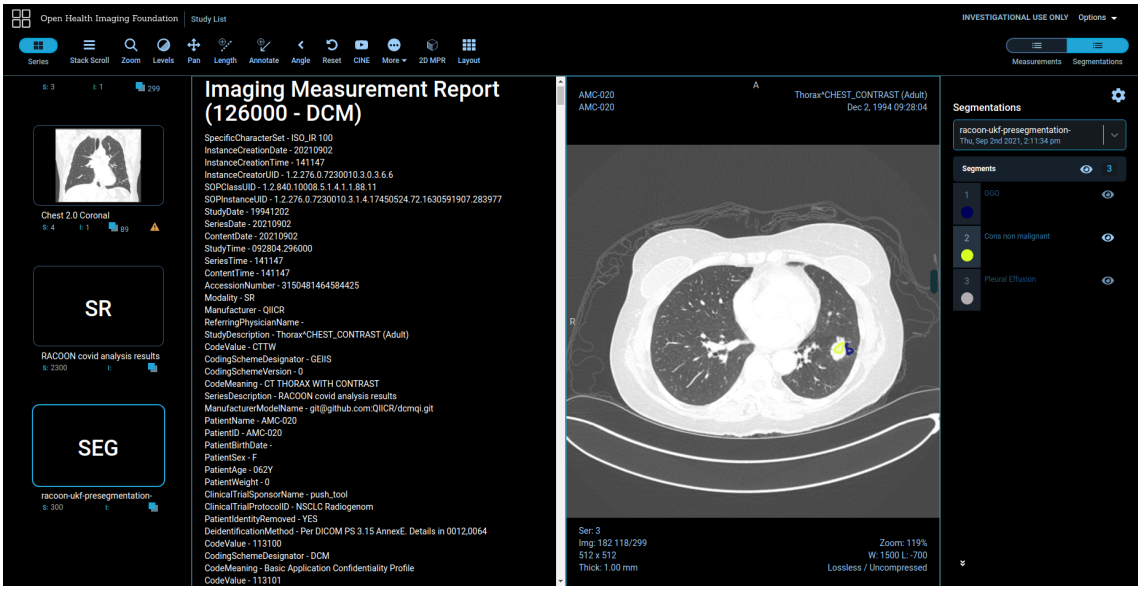

Figure 4.20: *OHIF showing the results of case scenario 2.4 with the DICOM SR on the left and the CT & DICOM SEG on the right.*

Although other processing pipelines have been developed externally, they will not be discussed in detail here due to the lack of variety of the processing steps besides the newly integrated method containers.

4.3.3 Use Case 3: Integration of Services and Desktop Applications

Beyond the integration of new algorithms, the possibility of additional or alternative services is also important for a platform. This includes both services that permanently offer additional features and software that can be started on demand to offer the service as part of a processing step within a pipeline. Since the platform's basic features are also implemented as such services, they could theoretically also be used for the evaluation of this use case. However, as these have already been examined several times elsewhere, the focus here was particularly on the integration of extensions, as this is also the most common way for partners to add new functionality to the framework.

Case Scenario 3.1: Permanent Services

The first case scenario addresses permanently running applications that are installed via the extensions mechanism and offer a web UI for user interaction. If the application has already been implemented as a web service following the client-server model, it

can be integrated very easily and quickly, as described in the "Integration of Services and Desktop Applications" section of the methods chapter (see 3.1.5).

Code-Server

For the evaluation, code-server [Coder, 2021], an integrated development environment (IDE) based on Visual Studio Code [Microsoft, 2021], has been examined first, since allows Operators and DAGs to be developed from within the browser and such directly on the platform server. Given that a container already exists for code server, it was just integrated into a K8s deployment and made accessible via a K8s service, which is accessible with the subpath "/code" within the platform. To make the folders containing the DAGs and Operators for FLOW available within the IDE, they are mounted into the container. Finally, a minimal Helm Chart had to be created as a package for the extensions.

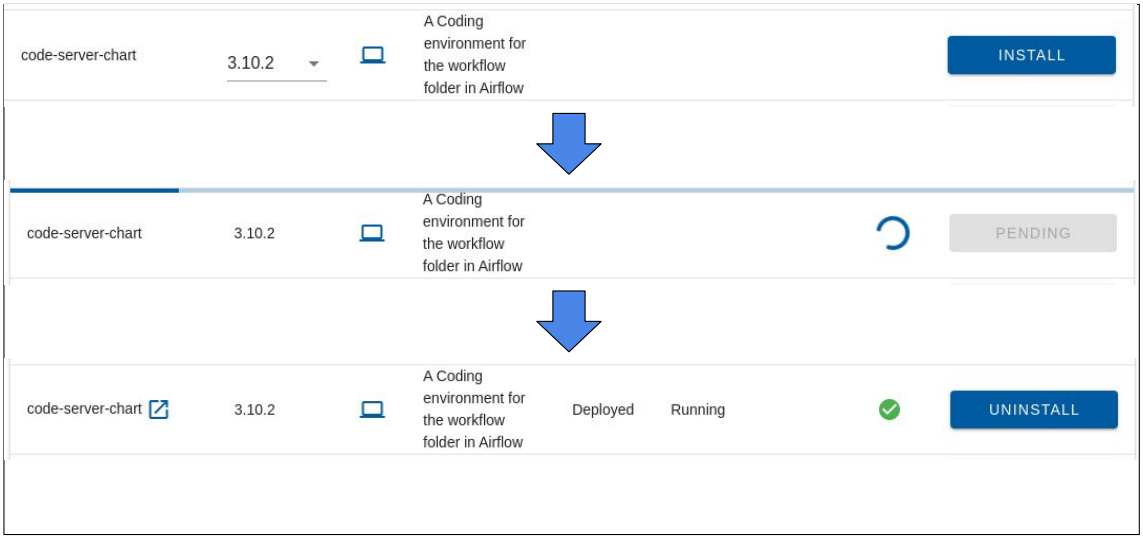

Figure 4.21: *Screenshot of the installation process for code-server*

After that, the extension is ready for installation and is listed in the corresponding section of the landing page, as shown in figure 4.21. After launching the installation process, the corresponding container is deployed and current process status is shown within the UI. Finally, the status "running" appears and the service can be accessed via the corresponding URL, which will open another browser-tab containing the services UI as seen in figure 4.22.

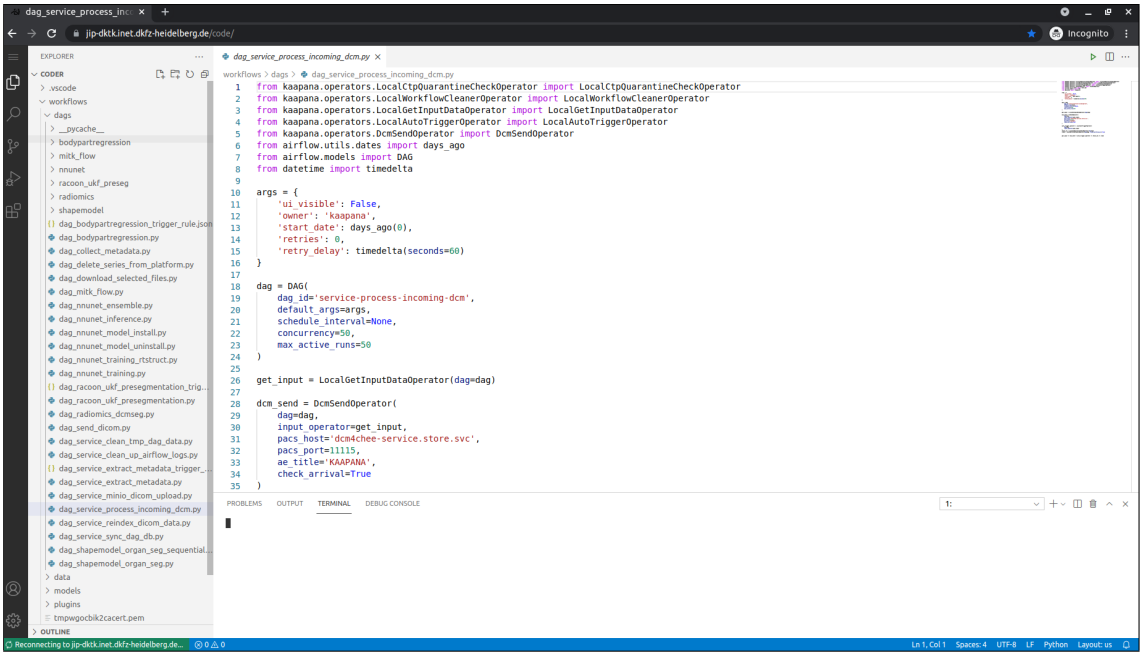

Figure 4.22: *Screenshot of Code-Server running in the browser.*

Tensorboard

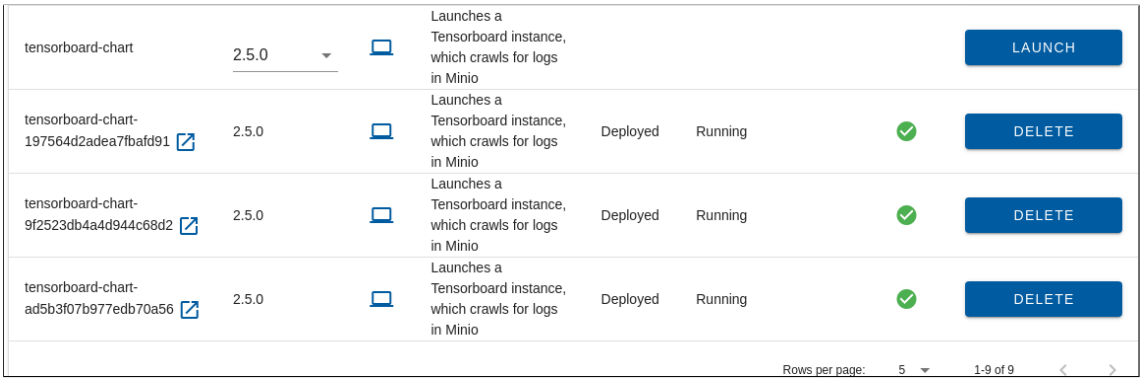

Figure 4.23: *Screenshot of the extensions with multiple instances running.*

As a second example, Tensorboard [TensorFlow, 2021] was integrated into the platform as a training monitoring tool, which is also already available as a web service and could be integrated just as easily as the IDE and installed via the extensions. The difference, however, is that this is a multi-launchable application that allows multiple instances of the service to run in parallel on the platform, accessible via automatically generated

subpaths. Figure 4.23 shows three running instances of the same Tensorboard application, which can be accessed individually via the link button. This is important if several instances of an application should be run in parallel, e.g. for the parallel creation of annotations or, as in this case, for the monitoring of various training processes. Figure 4.24 shows a screenshot of one of the running Tensorboard instances, which illustrates the current progress of a training session.

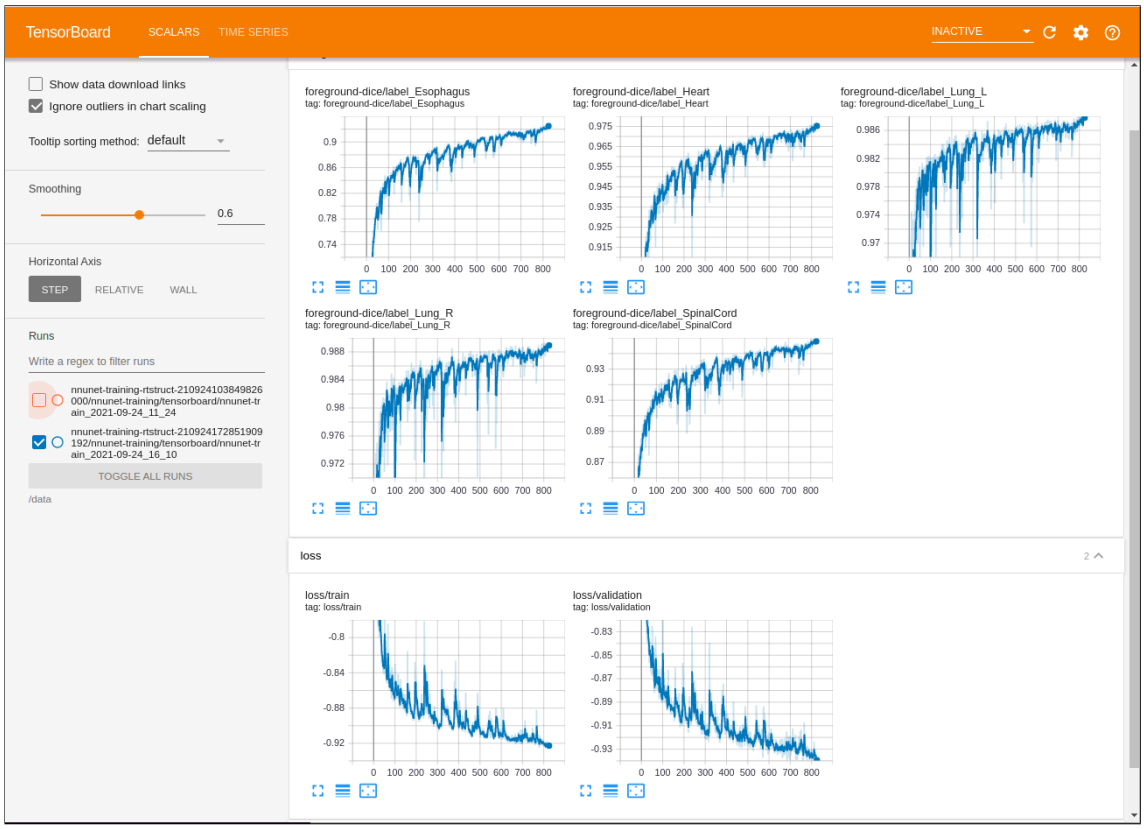

Figure 4.24: *Screenshot of Tensorboard visualizing a training of the nnUNet.*

MITK-Workbench

Finally, the MITK workbench has been used to also verify the integrability of desktop applications, as described in the "Integration of Services and Desktop Applications" section of the methods chapter (see 3.1.5). By using the provided VNC enabled base-container, the workbench UI could be made accessible via browser by installing a MITK binary release within the container. Data can be opened as usual through the "open-file" dialog of MITK, which allows access to the data stored in the object storage, which is mounted into the container.

Figure 4.25 shows the browser-based MITK Workbench showing a loaded CT scan and its corresponding segmentations from case scenario 2.2 in section 4.3.2.

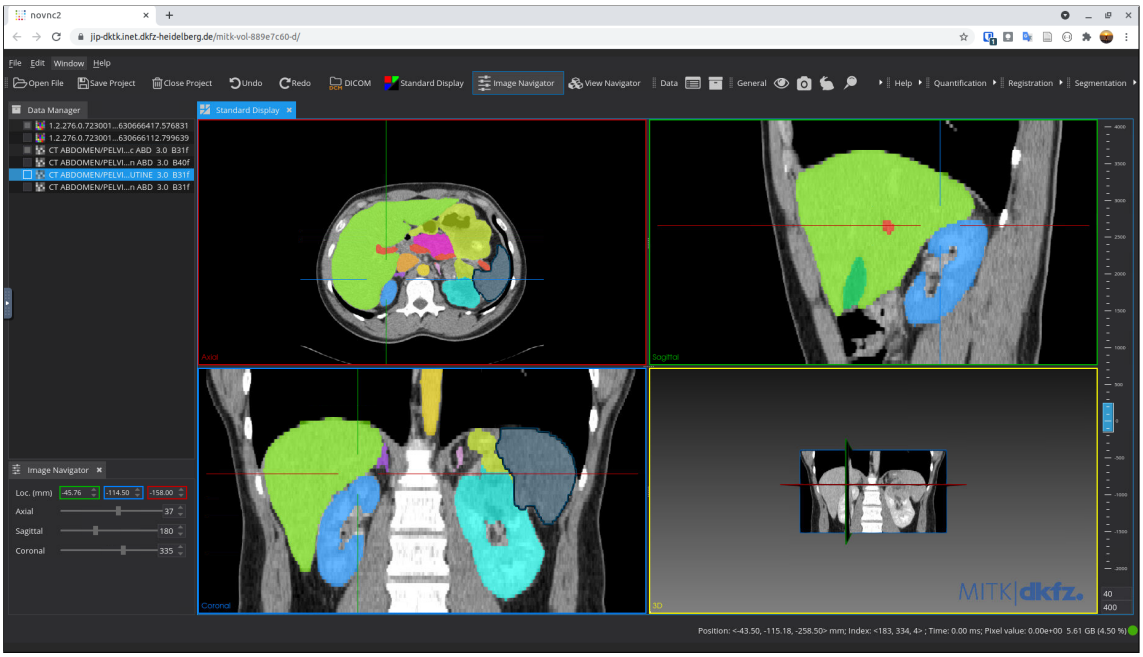

Figure 4.25: *Screenshot of the MITK-Workbench running in the browser.*

Case Scenario 3.2: On-Demand Services and Interactive Processing-Pipelines

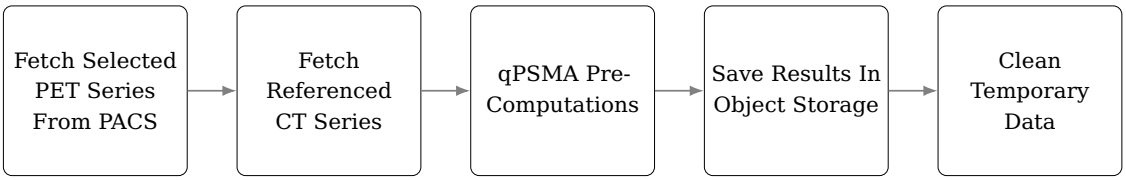

Figure 4.26: *Processing pipeline for the qPSMA pre-processing.*

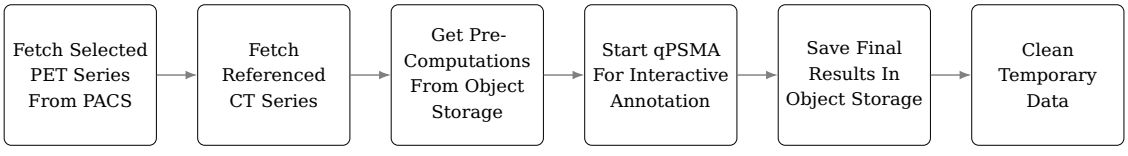

Figure 4.27: *Processing pipeline for the qPSMA interactive annotation.*

In this scenario, qPSMA, a semiautomatic software developed at the Department of Nuclear Medicine of the Technical University of Munich, which is used for whole-body tumor burden assessment in prostate cancer based on Prostate-specific membrane antigen (PSMA) using PET/CT imaging, was integrated into Kaapana.

The tool consists of a custom Python application, which uses its own UI to simplify the manual parts of analysis for the user. This involves time-consuming pre-processing of the data, that normally have to be triggered manually before the human interaction. During the integration, this step could be extracted so that it is automatically performed by the platform before the user starts the manual work, thus not having to wait for the results. Consequently, the analysis of the images involves both automatic and manual processing steps, which led to two challenges:

First, the custom Python application including the desktop UI had to be transferred via the provided VNC base-container, and furthermore, the extracted automatic pre-processing had to take place before it was actually deployed. Thus, as described in use case 2, for all incoming matching PET images the corresponding DAG can be triggered, which then automatically performs the pre-processing. After this is completed, a predefined number of qPSMA instances get started and wait for the user to manually annotate and process the data. On the landing page, the waiting instances are listed under pending applications as shown in the figure 4.28, and can be accessed via a link.

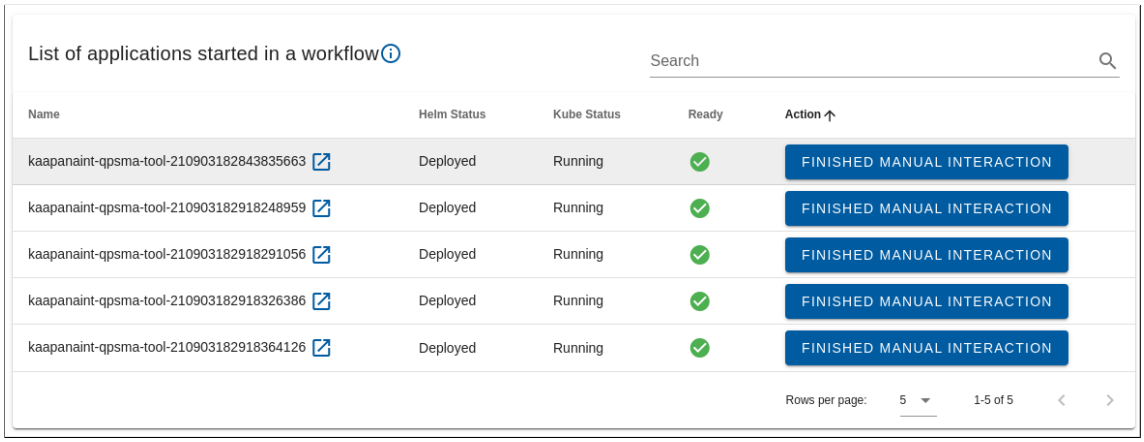

Figure 4.28: *Screenshot of the pending interactive processing steps.*

After the user has finished the work, pressing the button "Finished manual interaction" shuts down the respective instance, the results get saved in the object storage by the *LocalMinioOperator* and the next qPSMA instance gets launched and added to the list. Figure 4.29 shows the UI of qPSMA, which is used to perform the manual annotation.

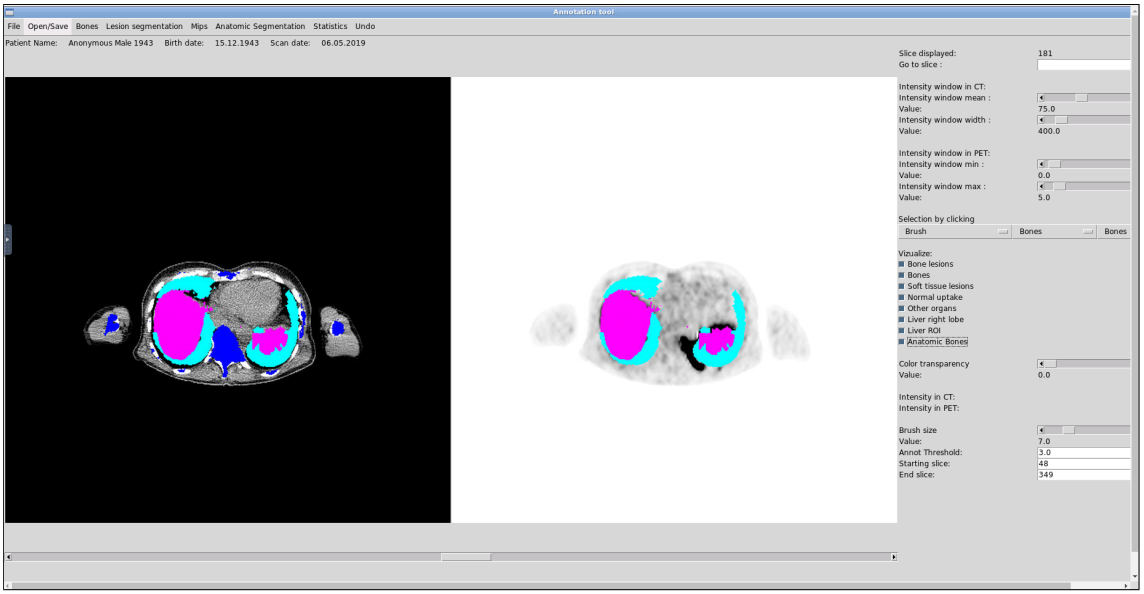

Figure 4.29: *Screenshot of the qPSMA application with its desktop UI in the browser.*

4.3.4 Use Case 4: Interactive Data Annotation

The fourth use case outlines the platform's capability for generating image annotations, which can then be used for the training of new models, or to generate data for studies. Here, the focus was again on segmentations, since these are central annotations in the field of MIC. The challenges mentioned in the use case description in section 3.1.5 have been addressed and implemented on multiple levels of the platform.

The problem of loading and saving the annotations and associated images had already been solved by the Operators provided. Here, the standardized conversion to DICOM guarantees that all relevant metadata is kept. Automatic predictions as described in case scenario 1.2 can be used here as pre-segmentation, with the result that the annotations only need to be manually verified or fine-tuned. For this manual step a customized version of MITK-Workbench has been developed, which focuses on segmentation and seamlessly integrates into the infrastructure. The workflow was implemented similarly to case scenario 3.2. Here, either an automatically generated segmentation or simply an image series is selected as input data for the DAG, which automatically downloads the data and also provides referenced series if necessary.

Then, the customized MITK workbench is launched as a manual interaction step and placed on the pending applications list available on the landing page. Figure 4.30 shows the customized MITK UI for segmentations, which automatically loads and

visualizes the corresponding image and pre-segmentation, if one exists. After the user has completed the review/refinement, a new DICOM SEG is generated and sent to the internal PACS afterwards the instance can be terminated via "Finished manual interaction" button.

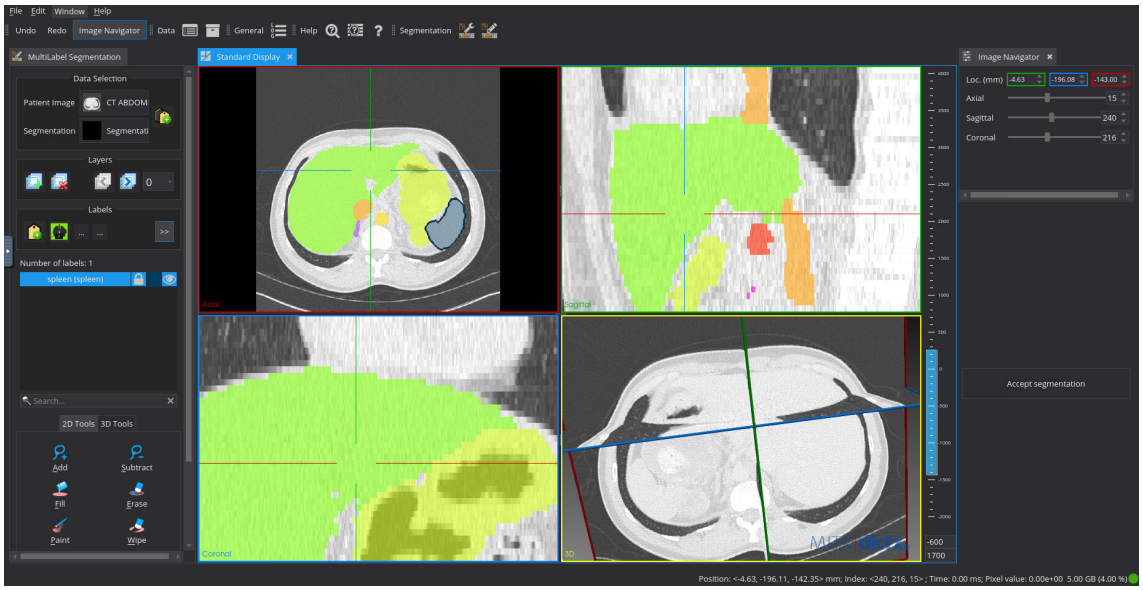

Figure 4.30: *Screenshot of the interactive MITK refinement application.*

4.3.5 Use Case 5: Machine Learning Workflows

The last use case deals with the training of methods using site-specific data in order to be able to generate new models. As in case scenario 1.2, nnUNet has been chosen for this task, as due to its automatic adaptation to the given data it is particularly well suited to offer a universal segmentation training method within the framework. Both local training but also federated scenarios are considered here, as is planned within the context of RACOON.

Case Scenario 5.1: Local Training

Local training uses data available locally on the platform to train a new model. This model can then be installed locally and used for inference on new data. It can also be shared with other partners so that they can make their own inferences using the model.

Input Data Selection

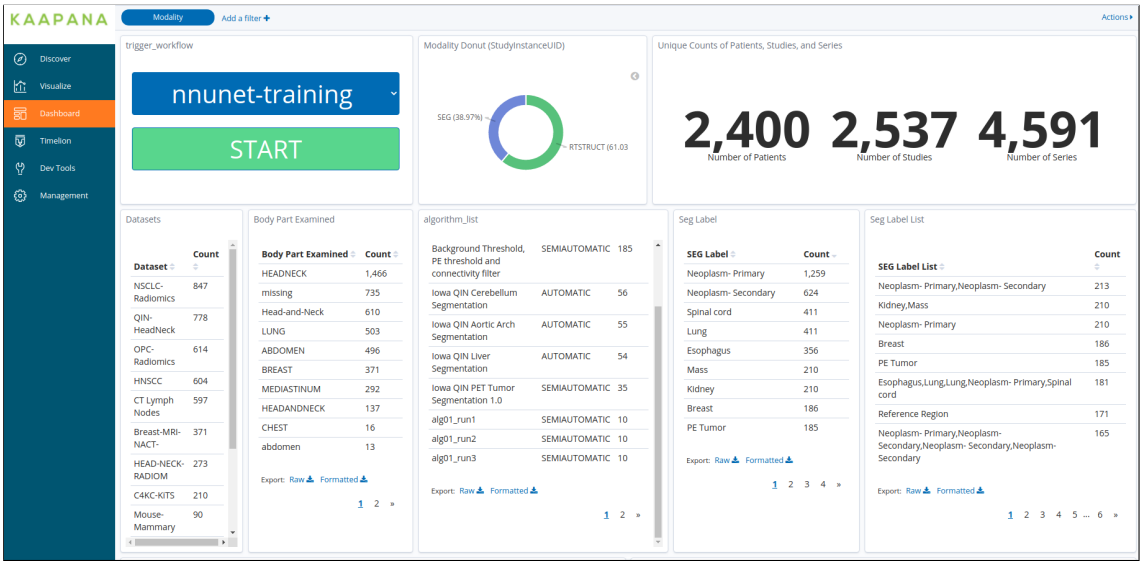

Figure 4.31: *Screenshot of the Segmentation Dashboard of META.*

Since nnUNet is specialized on segmentations, DICOM SEG or RT-STRUCT annotations are expected as input. To facilitate the data selection, a specialized segmentation dashboard has been added to META that pre-selects fitting modalities and provides detailed insights into the existing labels present.

Figure 4.31 shows a screenshot of this dashboard filled with sample data. For example, it is possible to retrieve all existing records that contain a particular label, even if they are both single-label and multi-label files that may contain different combinations of segmentation labels.

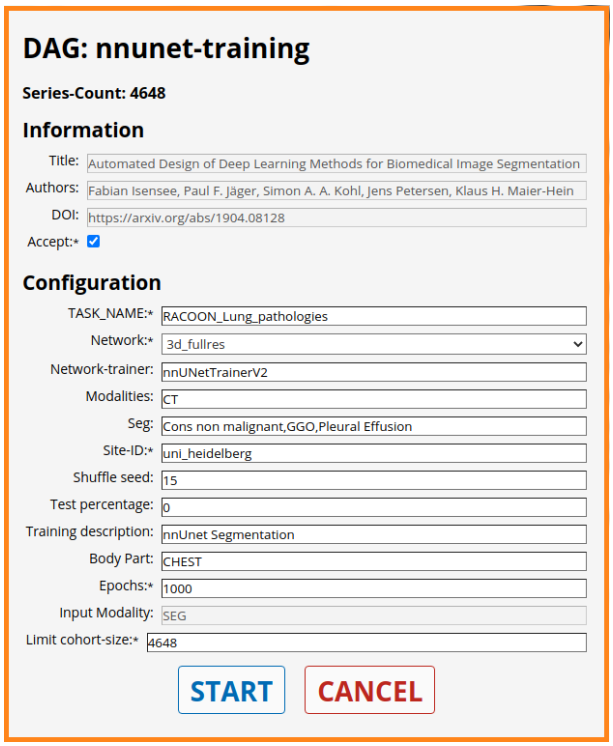

Figure 4.32: *Trigger dialog for the nnUNet-training DAG.*

Figure 4.32 shows the trigger dialog with all available parameters. For this training, a 3D full-resolution architecture was chosen, which should be well suited for learning the specified labels. Furthermore, the segmentation filters have been set, since only these structures should be learned, and thus all other labels occurring in the files should be excluded from the training. This allows white-listing of all labels to be included in the process. The site ID enables the subsequent mapping of models in the system to their corresponding sites - initially this ID is randomly generated, but can also be manually overwritten as in this case with "uni_heidelberg". Finally, the training can also be limited to a certain amount of epochs.

Processing Pipeline

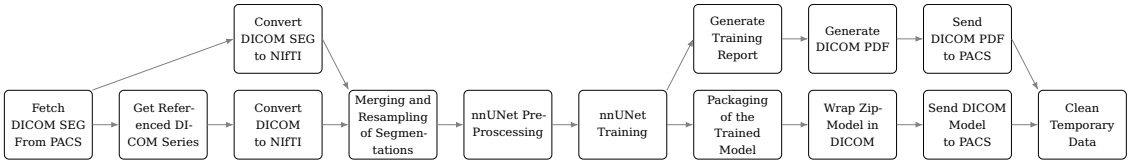

Figure 4.33: *Processing pipeline for the nnUNet training.*

The training-DAG mainly involves data retrieval and pre-processing to provide the data according to the specifications of the nnUNet. The first step fetches all RT-STRUCT and SEG series including its referenced source images from the PACS. Then all DICOMs are converted to NIfTI - Where during the conversion of multi-label segmentations are also separated as single files. Subsequently, by applying the segmentation filters, all non-whitelisted annotations are removed. After all DICOMs have been converted, the input data is ordered, verified and merged by the *SegCheckOperator*, which is used to

prepare the annotations for the training process. This pre-processing takes multiple steps, which are briefly described in the following:

1. **Data Organization:** Here, all annotations that are based on the same source series get collected and sorted. This is necessary since several segmentations based on the same source image may have been selected during the data selection process which could contain differing labels. Without this pre-processing, it would lead to issues in the training process, as for a single base image there would be multiple ground truths, each containing different labels.

2. **Alignment of the Label-Encodings:** Since different DICOM files may encode the same label differently, they have to be unified. After this step, it is ensured that labels are uniformly and incrementally encoded.

3. **Label Merging:** In this step, all existing label masks for a given source image are merged to create a single ground truth label map containing all associated annotations.

4. **Overlap Protection:** Since the nnUNet used in this example cannot handle multiple class labels for each voxel, it must be verified that there is no overlap between the different labels. In this processing step, this is inspected and, if necessary, corresponding annotations are rejected.

The result is a single NIfTI file for each source image, all of which use consistent label encodings without any mask overlaps. In the next step of the DAG, the nnUNet pre-processing is applied, which is based on *batchgenerators* [Isensee et al., 2020] and provides the final nnUNet specific training and test datasets.

Once these are ready, the actual training is launched, which automatically sets all necessary network parameters and runs as many epochs as specified during triggering. The progress of the training can be monitored using the Tensorboard extension described in case scenario 3.1. As shown in Figure 4.24, the progress of the individual estimated label DICE scores as well as the progression of the training loss can be tracked.

Results

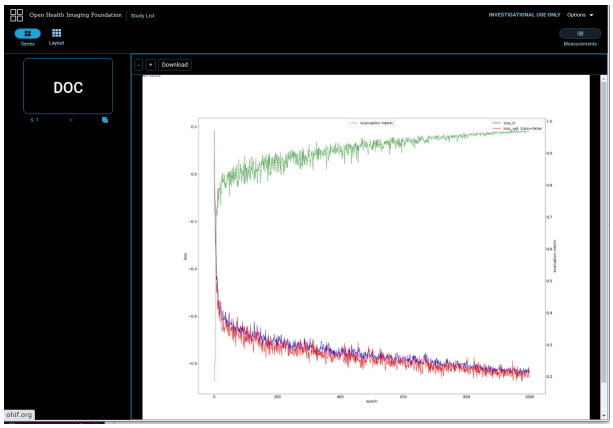

Figure 4.34: *Screenshot of the training report DICOM shown in OHIF*

After all epochs have been passed, a PDF report containing the training progress graphs of the training is generated and converted into a DICOM using the Encapsulated Document IOD. The result gets listed as DOC type within OHIF and the included PDF document can also be visualized in the viewer as it is shown in figure 4.34. In parallel, the trained model is extracted, while file whitelisting and pseudonymizing the included logs ensures that no potentially personal data such as concrete series UIDs of the images used for training is included in the model. The resulting ZIP-file contains the full DL-model and can be used for the prediction of images.

Since results should be managed as DICOM in Kaapana, the model is finally converted using the *Bin2DcmOperator* into a customized OT modality, which is then sent to the internal PACS. The META Trained Models Dashboard shown in Figure 4.38 lists all models available on the platform, where the "nnunet-model-install" DAG can be used for installation. Subsequently, the model is available for inference of upcoming acquisitions within the normal nnUNet-inference DAG. Since such models are DICOM objects, they can also be used with the other service DAGs, which allows, for example, the transmission to other Kaapana compatible instances or for model download via the integrated object storage.

Case Scenario 5.2: Federated Learning

The last scenario explores cross-site learning, which is intended to use the local data of all participants to potentially enable better predictions. The federated learning approach examined and implemented for this thesis addresses a training designed for RACOON, which, due to very limited external connectivity, utilizes an ensemble of multiple models. This procedure for training and subsequently combining multiple models involves two stages:

Model Training

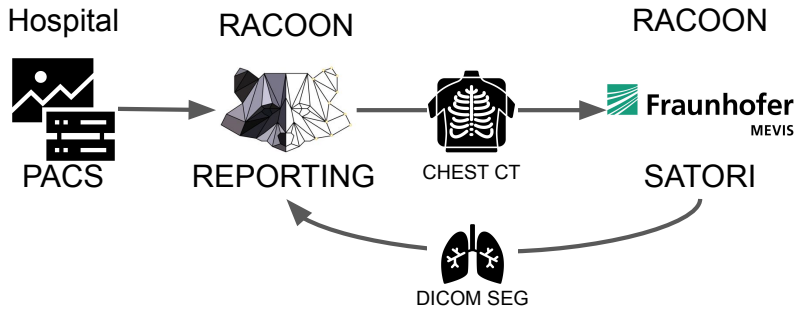

Figure 4.35: *Schematic representation of the RACOON segmentation workflow.*

The local training at the acracoon sites first requires the necessary annotations. The process established to accomplish this task is shown in figure 4.35. Here, the corresponding CT scans are first forwarded from the hospital PACS to RACOON REPORTING and then segmented with the help of SATORI. The annotations are conducted by the medical experts using a standardized diagnostic template, resulting in DICOM SEGs that include both anatomical structures and pathologically altered lung tissues that are commonly observed in the context of SARS-CoV-2 infections.

After the ground truth information is generated with SATORI, it is sent back to RACOON REPORTING where it is provided to the local JIP instance. Here, it is then used to train a new model, which is illustrated in figure 4.36. The basic procedure is similar to the approach of case scenario 5.1, using the nnUNet as well.

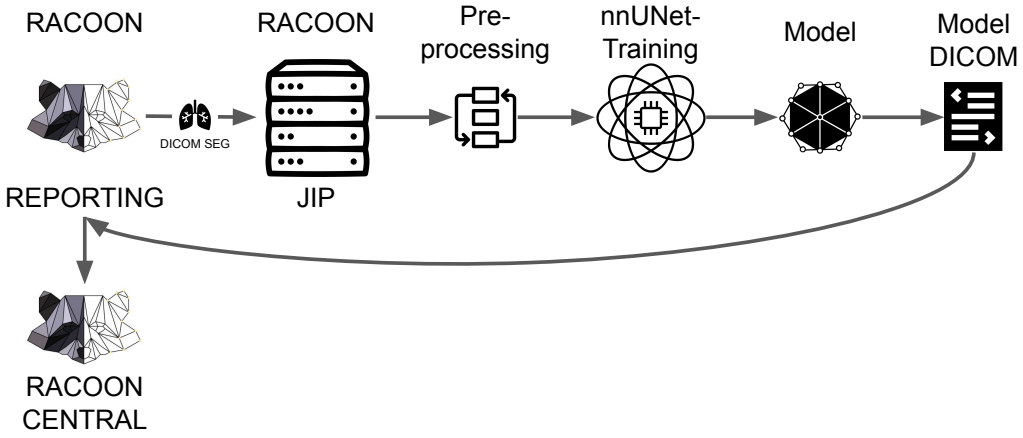

Figure 4.36: *Processing pipeline for the nnUNet-training for the RACOON use case.*

The result of the training is a site-specific model wrapped by DICOM, which is transmitted back to RACOON REPORTING. In order to combine and evaluate models from multiple sites into an ensemble, they must first be collected in RACOON CENTRAL.

For this transfer, a tunnel between REPORTING and CENTRAL must be established, which is the only external communication option available and therefore the only way of sending data out of the hospitals. Since this transfer imposes a size limitation, the model is not only converted into a single DICOM file, but rather split into several parts, which are related by a common series UID. This ensures that the transmission of data is feasible and the maximum chunk-size of the tunnel-transfer is not exceeded. This asynchronous execution of the training and model transmission to CENTRAL also allows the sites to operate independently of each other.

Model Ensemble

The second step of the distributed evaluation is performed on RACOON CENTRAL. After the DICOM models of multiple sites have arrived, a processing pipeline can be launched as shown in Figure 4.37, which evaluates and compares each of the models individually and as an ensemble. For a robust evaluation of model performance, it is important that a representative dataset of the hospitals is available on RACOON CENTRAL. This allows for a review of the performance of site-specific models on a dataset containing foreign data from other sites.

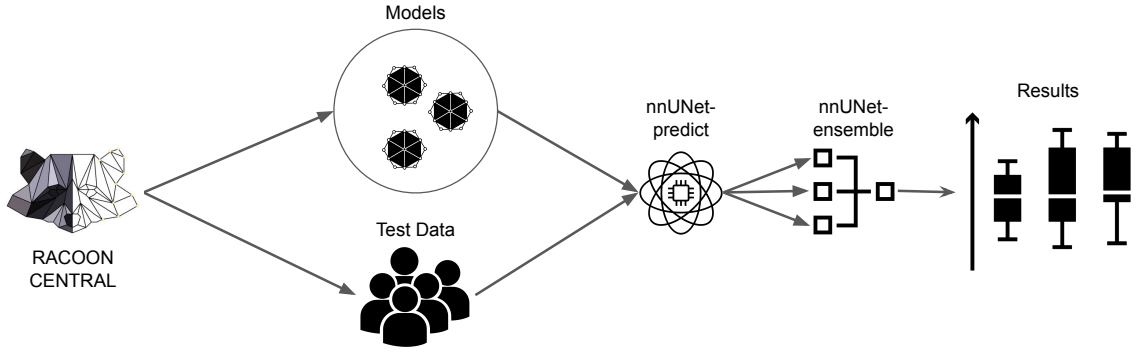

Figure 4.37: *Processing pipeline for the ensemble prediction of multiple models.*

Input Data Selection

The models to be combined and compared for the ensemble evaluation serve as input data. The data selection is also done using META by selecting the corresponding models on the Trained Models Dashboard shown in Figure 4.38.

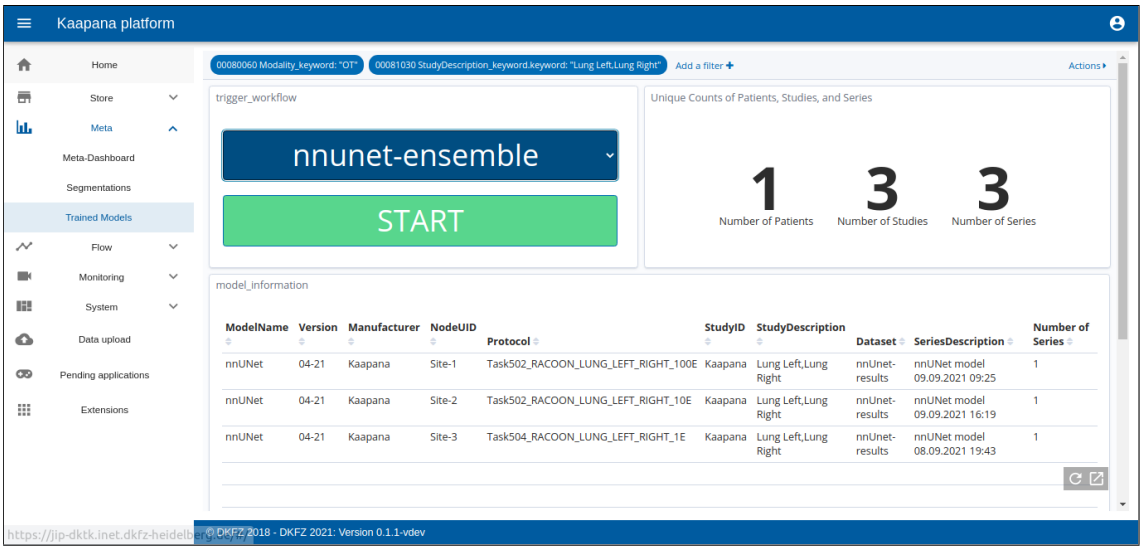

Figure 4.38: *Screenshot of the Trained Models Dashboard of META.*

After triggering the "nnunet-ensemble" DAG, the usual dialog is shown, which, in addition to setting the nnUNet parameters, also offers the possibility to whitelist labels to be considered for the evaluation. The test cohort is defined in advance using a fixed query in the pipeline.

Processing Pipeline

The DAG enabling this evaluation is complex, as it consists of many parallel operating processing steps as shown in Figure 4.39. It can be roughly divided into two sections, one responsible for extracting, assembling, and providing the models for the nnUNet, and the other obtaining and preparing the cohort data for analysis from the PACS.

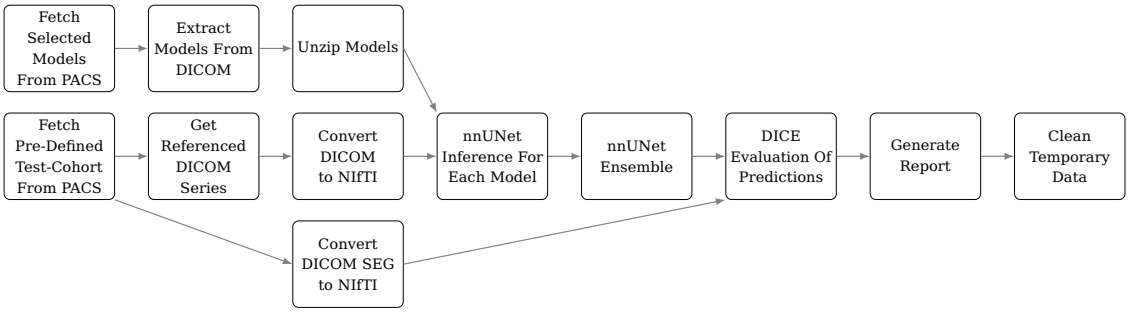

Figure 4.39: *Processing pipeline for the ensembling of models.*

First, the chunks that were originally generated from the binary models need to be extracted from the selected DICOM files and re-assembled so that the complete models are available for predictions of the nnUNet. In parallel, a predefined test-cohort dataset is fetched from the PACS, which includes the ground truth segmentations as well as the CT scans.

For the comparison of all models, a prediction is generated from each source image using each of the selected models. Besides the predictions of the individual models, a combined prediction of the ensemble is also produced by majority voting, which is also included in the comparison. Finally, all generated predictions are compared to the ground truth to determine the corresponding DICE scores for each label, model, and ensemble.

Results

As a result, this evaluation generates various graphs and tables that produce a performance comparison of the different models and the ensemble on all test data.

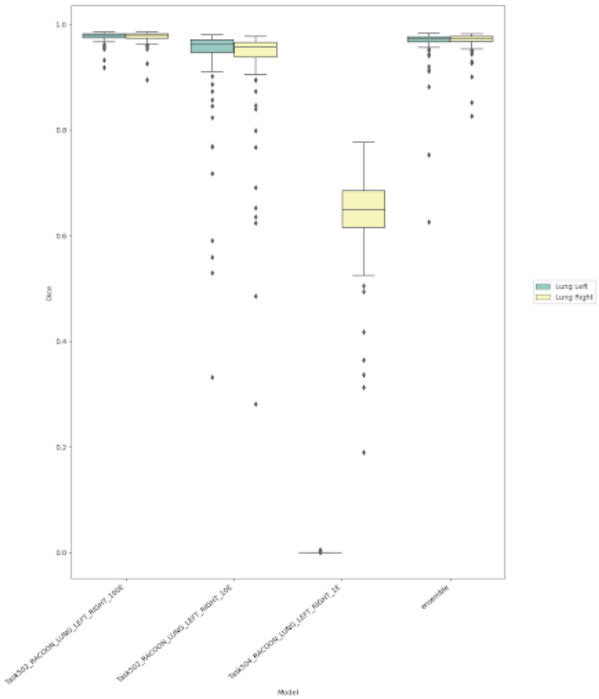

Figure 4.40: *Box-plot of the results for the model ensemble.*

Figure 4.40 outlines an example plot where three separate models, which were trained with the NSCLC Radiomics dataset [Aerts et al., 2019]. The first model on the left was trained with 100 epochs , the second with 10 epochs and the last with only one epoch. As expected, the graph indicates decreasing performance of the models with decreasing epoch count and the still solid performance of the ensemble on the right. All results are provided both as images as well as raw data in JSON files. The pipeline described here already shows the complete implementation to be used in RACOON. At the time of writing, it was unfortunately not yet feasible to run experiments across the consortium, as not all the necessary data had been annotated and CENTRAL, including the test data, was not yet available.

5 Discussion

Clinical evaluation, training, and application of new data-driven image analysis methods requires a sophisticated digital infrastructure that allows new techniques to be deployed within real-world clinical environments. For this purpose, technical concepts have been developed that address the various challenges involved in the development and establishment of such an infrastructure. Since projects in the context of medical imaging are very diverse, the aim was not to have a single platform covering all potential applications, but rather to provide a flexible and modular framework for the construction of project-specific platforms that can then be tailored to meet specific project requirements. A high degree of modularity and extensibility should also enable usability for future use cases that have not been considered in so far. The evaluation has shown that, overall, the concepts were effective and the use cases could be implemented using the framework tools outlined.

The real practicality of the implementations, on the other hand, was difficult to verify, since this thesis includes the initial design and first implementation of the software, and thus most of the real world deployments have just started at the time of writing. Although it could be shown that the servers in the clinics could be put into operation and also updated with the developed solutions, long-term operation and stability of the platforms cannot yet be conclusively assessed. Even though the projects have not been on going for very long, some conclusions can already be drawn for the main sections of the system:

5.1 Data Accessibility, Exploration and Cohort Selection

In this context, the primary focus was on DICOM for the storage and communication of image information, as this was chosen as one of the central features of the platform. This choice has also proven to be successful in the hospitals, as overall only few problems have occurred in practice during interaction with clinical images and PACS. As far as is known, all of the participating university hospitals were able to transmit their data from the hospital PACS to the platform via the usual local transmission mechanisms. Problems only occurred in the context of preceding anonymization of the data, as this occasionally hindered the referencing of the data with the corresponding annotations. In the case of pseudonymization or anonymization procedures, it is important to ensure that the references are adopted and changed in a uniform manner. However, it also became clear that annotations in DICOM are not yet widely used, and some of the collaborators needed to implement the functionality first to be compatible with the platform. This also implies that many of the already existing annotations cannot currently be used in the platform, as they are only available in scientific data formats such as NIfTI or Nrrd, which do not include the required reference metadata. In the future, an additional upload option will allow such data to be used in the platform by manually providing the required metadata so that the platform can generate the appropriate DICOM-compatible files. This is also particularly interesting to be able to utilize existing challenge datasets within the platforms.

On the other side is the metadata, which is responsible for data exploration and selection within META. In this regard, the utilization of the technology stack with ES and Kibana provided a fast and pragmatic way to meet the basic requirements of the metadata system. Especially the visually attractive dashboards, which can be dynamically customized and created by the user, offer great benefits. Yet it is also apparent that the technology was originally developed for processing logging data and thus will not meet the requirements of this system in the midterm. The limitations here come mainly from the way ES manages and stores the data, as it does not handle nested data structures well. However, the DICOM JSON model, which enables the standardized conversion of metadata into JSON, contains a lot of deep nesting, which has led to a flattening of the metadata representation in Kaapana. Furthermore, it has been shown that large ES indexes are associated with performance losses, so that it was chosen not to index the metadata of each individual DICOM objects, but rather to prefer a series-based indexing.

Given that META in the current data model is merely an additional view of the data stored in the PACS, it is also not suitable for storing textual processing results, which would often be convenient in practice to have this additional information available for

data exploration and selection. In the future, the implementation of META should be replaced by a system that retains the core concept of DICOM metadata, dashboards, and visual cohort filtering, but with a different technological foundation that allows object-based indexing and extensibility with supplemental information. Furthermore, by linking other hospital systems such as the HIS, RIS or Laboratory Information Management System (LIMS), it would also be conceivable to expand META by laboratory findings, medical reports or insurance information. It would also be interesting to be able to collect imaging metadata from the hospital PACS so that the information is available within META, but without the need to actually keep the image data on the analysis servers. Which would allow the data to be downloaded from the corresponding systems on demand only, which, thanks to the already existing DICOM-based data fetching, would only need to change the source system for the data. However, this would also require the analytics server to have access to the central clinical image archive, which would also raise security and privacy challenges.

Another major problem in the context of metadata is the already described non-standardized content of some header entries. This applies to both images with e.g. series, study or protocol descriptions and annotations such as label identifiers. Kaapana includes initial steps to improve this, for example by predicting the body part examined of the images, or by determining the labels of the generated annotations based on SNOMED CT. However, these problems can only be properly solved by either introducing standardized values, as has been accomplished in context of the RACOON project for lung specific imaging, or through progress in predicting metadata based on the raw image information. Kaapana offers the possibility to apply such methods automatically and to make the obtained information accessible. Moreover, there is also the possibility to cross-check the existing metadata with the predicted information.

Another known limitation of the current data concept is related to data separation for projects or user groups, which is currently inadequate. The platform's access restriction currently only applies to entire areas or specific services of the infrastructure, but not to individual data records within them. During the work in the projects and consortia, however, it quickly became apparent that a deeper separation of data, as well as access control, is desirable, since not every user of the platform should have access rights to all project data for data privacy reasons. To circumvent this problem, it is possible to set up a separate platform instance for each project, which is then only accessible to certain groups of people. This can also be achieved by simply swapping the underlying file system for each project, with the limitation that only one project at a time can work with the platform. But this is only a workaround and should be solved in the future by adequate internal data separation.

5.2 Algorithmic Accessibility and Data Processing

The concept of data processing has overall proven to be efficient and so far, all requested methods could be successfully integrated into the framework. This applies both to the provisioning of the resources required by the methods in the form of computing power and compatible input data, as well as on the flexible adaptability of the concepts to suit existing implementations as far as possible. It is also important to note that community partners who are not part of the core development team have managed to integrate their own methods independently, indicating that the concepts are feasible for non-infrastructure experts. This is at least true for typical use cases where images serve as input - however, experience with other applications that work with other media such as text or lab information is very limited. Here, more testing and experience in the future will have to show whether the mechanisms also work for other types of data processing. Certainly, the more complicated the processing pipeline becomes, the less likely it is that it can be implemented with simple abstractions such as those offered in Kaapana. For these cases, however, there is always the possibility to omit the offered simplifications and to implement a custom solution. An unresolved problem remains when multiple non-standardized input modalities (such as multiple MRI research protocols) are used for each case to be processed, as identification of these related series is difficult without a reliable label. This could be solved by manually configuring a mapping between local and standardized labels.

The concept of data processing consisting of the interaction of META with data selection and parameter specification together with FLOW for the management of the actual execution of the jobs, is quite basic but functional. The introduction of pipeline-specific dynamic parameter dialogs has proven to be very powerful, as it allows many options to be set before the processing pipeline is triggered. For the visualization and control of FLOW, the standard UI of Airflow has been used for the most part, which is very technical and requires a lot of expertise to comprehend. Frequent request and feedback from many partners who used the platform for the first time revealed that it is too complex and non-intuitive for many users. To improve this, an additional user interface would be useful, providing experiment management and abstracting the underlying processes in terms of DAGs. A simple view of the running processes with basic functions like start, stop or reset of an experiment, would already be an improvement. Additionally, separating cohort management from experiment setup would be desirable, as it would allow multiple experiments to be conducted with the same dataset or better tracking of the data used for each experiment. In principle, this is already possible, but there is no easy access to these functions and information.

Training of new segmentation models integrated with nnUNet shows how powerful

such method integration can be. Almost any segmentation annotation on the platform can be selected to train a new state-of-the-art model without having to understand anything about the technology or the need to make any specific adjustments to data. Fully automated data pre-processing, sorting and merging allows for the first time a one-click creation of models for segmentation tasks. The wrapping in DICOM allows these models to be easily transmitted, downloaded and distributed between the participating partner sites. The next level of clinical image processing is the distributed training of models with real data from clinical routine. So far, research in this area has mainly relied on simulations, as it has not been possible to conduct such experiments in the real world due to the high technical and organizational hurdles. Kaapana can now help to overcome these difficulties by providing a unified processing environment for distributed scenarios. The approach evaluated in this work represents one of the simplest flavors of distributed learning that uses an ensemble of several separately trained models. It was chosen because it does not require a constant connection between the clinical instances of the platform and training can be performed in an isolated and asynchronous manner. Within RACOON, for which this method was implemented, there also exists a central instance, JIP CENTRAL, which can be used for the collection and combination of the different models. The models can be transmitted from the clinics to the central instance via the existing DICOM route. This also highlights the current greatest challenge associated with true distributed learning between clinics: Connectivity.

While isolation offers great advantages for data protection, it creates challenges for training approaches that require constant communication between partners. Currently, there is no way in the German consortia to enable such real-time communication between the main instances - an alternative would be additional platform instances that are operated in the demilitarized zone with the purpose of enabling such scenarios. Initial collaborative efforts indicate that both trust and willingness to enable such practices are clearly increasing. The commitment to enable such a connection has already been signaled by several partners, so it is probably only a matter of time until the first synchronously distributed models will be trained.

5.3 Server Installation and Integrability

The establishment and setup of the servers at the hospitals has generally worked quite well. Challenging was the fact that the entire installation procedure had to be handled by local technicians who did not necessarily have expertise in the given technologies. This resulted in the need to optimize, automate and simplify the whole process from the installation and configuration of the operating system up to the running platform. Since scalability is a key characteristic of the entire initiative, this time-consuming process was worthwhile as it enabled the platform to be installed and deployed independently. Furthermore, such automations as the build system and installation scripts facilitate developers' work with the platform.

One issue that was frequently encountered during installation was the need for external registry access, which is required for the installation process. It turned out that even temporary Internet access during the installation posed challenges for the local IT teams at the hospitals, as the local proxy servers had to be explicitly configured to provide access for every single external resource that was requested during the installation. However, until now, processes have not been optimized to minimize these sources, resulting in significant additional overhead for local technicians. In the future, this could be facilitated by providing central mirrors not only for the container registry, but also for other system software, such as the CentOS or Ubuntu package repositories, so that only one target server is required for the entire installation process. An alternative approach, as taken with RACOON, is the distribution of entire VMs to hospitals, which already contain all the necessary components. However, this has other drawbacks in terms of fast upgradability and extensibility or virtualization overhead, which makes a case-specific consideration necessary.

The integration of the servers into the clinical IT landscape itself is currently limited to the ability to send images from the local PACS to the platform. For the transfer of images, this approach has proven to be very effective, simple and pragmatic, as it eliminates the need for access from the analysis server to the clinical systems. For the future, it would also be desirable to have the information from the other clinical data systems such as HIS, RIS or LIMS available for data processing in Kaapana. Here, HL7 would be a good choice to implement as an additional interface into the framework, as it is offered by many other clinical systems.

5.4 Security and Data Sovereignty

Due to the isolated operation of the servers in strictly controlled IT environments, security was not a central topic of this thesis. So intended malicious attacks from the inside, such as those that could be caused by malicious third-party processing steps, are currently not well shielded by the framework and the platforms based on it. This work is based on the assumption of cooperating, known partners sharing their methods, which are received, integrated, and deployed centrally by trusted parties. The collaboration within the consortia described are based on mutual trust. For a future opening of the extension catalog and the associated acceptance of foreign add-ons from possibly unknown third parties, an extended security concept is required that identifies and ultimately eliminates all possible data processing attack vectors. This can be achieved, for example, by isolating the third-party applications in a sandbox, which by design would automatically prevent access to all data that is not required. Furthermore, some kind of access control policy with explicit permissions, such as those used by modern smartphone operating systems and their apps, would be a potential approach to reduce the risks. Sophisticated new techniques such as Network Behavior Anomaly Detection (NBAD) could also help to detect misbehaving systems. However, since this would have exceeded the scope of this thesis, this subject is explicitly addressed in specialized projects, such as Trustworthy Federated Data Analytics (TFDA) [CISPA, 2021] and the achievements of them will be incorporated into Kaapana's code base in the future to also enable secure and trustworthy execution of third-party analyses in the framework as well.

Nevertheless, at least the basics and best practices for implementing a container-based infrastructure were followed, and the concepts were carefully chosen to ensure secure operation. Thus, the isolated server operation prevents any data from leaking out. With CentOS and Ubuntu, two modern operating systems are supported that are already used in many data centers and can be kept up to date with regular security updates. Likewise, the containers being used and their base images should regularly be scanned and reviewed for known security vulnerabilities. For the JIP and its dedicated container registry, this process has already been implemented by automatically scanning all containers for vulnerabilities every night.

Another critical question is how secure container technology is in general, since it is ultimately based on the host's kernel and therefore does not provide the level of isolation that VMs do, for example. Given the widespread and broad industrial application of this technology, however, it can be assumed that these, albeit rather theoretical, issues will also be solved in the future. With virtualized containers, execution in the user namespace, and the use of non-root users within the containers, there are already

ways to minimize the risks associated with a container escape. Likewise, using the established OCI standard for containers ensures that new developments, such as more secure container runtimes like Kata [Kata, 2021] or CRI-O [Cloud Native Computing Foundation, 2021], can be easily adopted and substituted in the future.

6 | Summary

The emergence of new data- and algorithm-driven analysis methods is revolutionizing many areas of research and enabling solutions to problems that were previously considered intractable. But does this also apply to medicine?

Improving diagnosis, fine characterization of patients for personalized medicine, monitoring disease progression and its prognosis, or predicting the outcomes of various therapies are just some of the areas that could potentially benefit from such new analysis techniques based on Deep Learning, which has enabled major advances in computer vision and related fields. Such algorithms now enable machines to better understand and interpret visual image data, which on the one hand offers promising perspectives for medical image processing, but on the other hand also poses challenges. As such, large amounts of annotated data are needed to prevent overfitting and to generate robust, generalizing and reliable models. Multicenter imaging studies could greatly improve the availability of such data and also enhance heterogeneity by obtaining training data from various sites. However, sharing data across multiple sites has proven to be difficult due to the high level of data protection associated with medical records and technical challenges such as interoperability. Consequently, this thesis attempts to avoid the necessity of data exchange by following a different approach:

"Let’s share the algorithms, not the data!"

The objective and central research question here is whether it is possible to shift the evaluation and training of modern image analysis methods to the clinical data owners and how this can be accomplished. Although this approach helps to circumvent the data export and the associated data protection challenges, the participating partners must still be enabled to execute the algorithms from a technical and organizational perspective. For this purpose, this dissertation investigated concepts, the development and establishment of a decentralized infrastructure for clinical medical image analysis that enables standardized data access and uniform execution of analysis methods for joint data analysis in the context of multicenter imaging studies.

The technical realization of this infrastructure was achieved by developing a software framework called Kaapana, which enables the building of imaging platforms. The resulting software can be hosted on dedicated servers within the clinical IT environment to be operated isolated from any external connectivity and to be interconnected with other local clinical systems. By leveraging modern cloud technologies such as containers and Kubernetes, the local deployment provides a private cloud for image processing that can be accessed from any locally connected workstations via the web browser. Standardized linkage to the clinical PACS via DICOM and the integration of a research imaging archive enables redundant data management and consistent data access for the execution of analysis methods. Dashboards enable efficient data exploration and filtering by visually presenting DICOM metadata of the platform's data and allowing it to be selected via search queries. A uniform execution environment for data processing allows algorithms to be applied to such selected data and to be uniformly packaged and shared with partners. High-performance server hardware including Graphics Processing Units enables a variety of analysis techniques such as Deep Learning-based model inference, as well as the training of new models to be shared with partners. Within the infrastructure, the standardized and widely adopted DICOM format has been prioritized so that also many analysis results can be provided in a standard-compliant way. Using formats such as DICOM SEG, SR or Encapsulated PDF, data annotations become compatible with clinical workflows and IT systems.

With support of the Radiological Cooperative Network and the German Cancer Consortium, the resulting infrastructure could be deployed and evaluated within two German research consortia. To this end, all 36 German university hospitals have commissioned their own server on which the platform has been installed and tested. This involved evaluating the commissioning, integration, operation and maintenance of such a decentralized network, as well as various use cases designed to cover the typical tasks of such a system. As a result, these use cases and the corresponding varying demands could be realized with the developed concept and the implemented framework. A flexible extension mechanism also allows the integration of additional services such as

the MITK workbench or processing algorithms such as the nnUNet into the framework. Furthermore, first analysis pipelines developed by external partners could also be integrated and delivered to the partners already.

Within the scope of this work, clinics were enabled to apply up-to-date research methods to their own data through the development, distribution and support of the developed infrastructure. Since the research consortia, which already include all German university hospitals, have only just started their activities, there are great opportunities to make the high-quality data from the partner sites accessible and usable for research in the future. Because of Kaapana's open source code, its architecture based on common industry standards, and its already broad deployment, this framework could also serve as a foundation for areas other than medical imaging, and thus offer the potential for tighter data integration for clinical computing in general.

7 Zusammenfassung

Das Aufkommen neuer daten- und algorithmengestützter Analysemethoden revolutioniert derzeit viele Forschungsbereiche und ermöglicht Lösungen für Probleme, die früher als unlösbar galten. Aber gilt dies auch für die Medizin?

Die Verbesserung der Diagnose, die Feincharakterisierung von Patienten für die personalisierte Medizin, die Überwachung des Krankheitsverlaufs und seiner Prognose oder die Vorhersage der besten Therapie sind nur einige Beispiele, die potenziell von solchen neuen Analysetechniken profitieren könnten. Algorithmen ermöglichen es Maschinen nun, visuelle Bilddaten besser zu verstehen und zu interpretieren, was einerseits vielversprechende Perspektiven für die medizinische Bildverarbeitung bietet.

Andererseits aber auch Herausforderungen mit sich bringt, da große Mengen an annotierten Daten benötigt werden, um ein Overfitting zu verhindern und somit robuste, generalisierende und zuverlässige Modelle zu ermöglichen. Multizentrische Bildgebungsstudien könnten die Verfügbarkeit solcher Daten erheblich verbessern und auch die Heterogenität durch die Zugänglichkeit von Trainingsdaten aus verschiedenen Standorten erhöhen. Die gemeinsame Nutzung von Daten über mehrere Standorte hinweg hat sich jedoch aufgrund des hohen Datenschutzniveaus, das mit medizinischen Daten einher geht und technischer Herausforderungen, wie der Interoperabilität, als schwierig erwiesen. In dieser Dissertation wird daher versucht, die Notwendigkeit eines Datenaustauschs zu vermeiden, indem ein anderer Ansatz verfolgt wird:
"Lasst uns die Algorithmen teilen, nicht die Daten!"

Ziel und zentrale Forschungsfrage ist dabei, ob die Evaluation und das Training moderner Bildanalyseverfahren in die Kliniken verlagert werden kann. Dieser Ansatz kann zwar den Datenexport und die damit verbundenen datenschutzrechtlichen Herausforderungen umgehen, allerdings müssen die beteiligten Partner auch technisch und organisatorisch dazu in der Lage sein, diese Aufgaben zu erfüllen. Zu diesem Zweck wurden in dieser Dissertation Konzepte, sowie die Entwicklung und Etablierung einer dezentralen Infrastruktur für die klinische medizinische Bildanalyse untersucht, die einen standardisierten Datenzugriff und eine einheitliche Umgebung für die Ausführung von Analysemethoden im Rahmen von multizentrischen Bildgebungsstudien ermöglichen.

Die technische Realisierung dieser Infrastruktur wurde durch die Entwicklung eines Software-Frameworks namens Kaapana erreicht, welches den Aufbau von technischen Plattformen für die Bildanalyse ermöglicht. Die resultierende Software kann auf dedizierten Servern innerhalb der klinischen IT-Umgebung betrieben werden, um isoliert von jeglicher externer Konnektivität betrieben und mit anderen lokalen klinischen Systemen verbunden zu werden. Durch die Nutzung moderner Cloud-Technologien wie Containern und Kubernetes bietet die lokale Bereitstellung eine private Cloud, auf die von den klinischen Arbeitsstationen über den Webbrowser zugegriffen werden kann. Die standardisierte Anbindung an das klinische PACS über DICOM und die Integration eines Forschungs-PACS ermöglichen eine redundante Datenhaltung und einen konsistenten Datenzugriff für die Durchführung von Analyseverfahren. Dashboards ermöglichen eine effiziente Datenexploration und -filterung, indem sie die DICOM-Metadaten visuell darstellen und über Suchanfragen selektierbar machen. Eine einheitliche Ausführungsumgebung für die Datenverarbeitung ermöglicht es, Algorithmen anzuwenden und sie einheitlich zu verpacken und mit Partnern zu teilen. Leistungsstarke Server-Hardware, einschließlich Grafikkarten, ermöglicht eine Vielzahl von Analyseverfahren wie Deep Learning-basierte Modellinferenz sowie das Training neuer Modelle. Innerhalb der Infrastruktur ist DICOM das grundlegende Datenformat, so dass auch viele Analyseergebnisse standardkonform bereitgestellt werden können. Durch die Verwendung von Formaten wie DICOM SEG, SR oder Encapsulated PDF werden auch Annotationen mit klinischen Arbeitsabläufen und IT-Systemen kompatibel gemacht.

Mit Unterstützung des Radiological Cooperative Network (RACOON) und des Deutschen Konsortiums für Translationale Krebsforschung (DKTK), konnten zwei große deutsche Forschungskonsortien für die Evaluierung gewonnen werden. Dabei haben inzwischen alle 36 Universitätskliniken in Deutschland einen eigenen Server in Betrieb genommen, auf dem die Plattform installiert und getestet wurde. Hierbei wurden die Inbetriebnahme, die Integration, der Betrieb und die Wartung eines

solchen dezentralen Netzwerks sowie verschiedene Anwendungsfälle evaluiert, die die typischen Aufgaben eines solchen Systems abdecken. Im Ergebnis konnten diese Anwendungsfälle und die damit verbundenen unterschiedlichen Anforderungen mit dem entwickelten Konzept gut realisiert werden. Ein flexibler Erweiterungsmechanismus erlaubt zudem die Integration von zusätzlichen Diensten wie der MITK-Workbench oder Algorithmen wie dem nnUNet. Darüber hinaus konnten auch bereits erste von externen Partnern entwickelte Analysepipelines integriert und ausgeliefert werden.

Im Rahmen dieser Arbeit wurden die Kliniken durch die Entwicklung, Etablierung und Unterstützung der entwickelten Infrastruktur in die Lage versetzt, aktuelle Forschungsmethoden auf ihre eigenen Datenbestände anzuwenden. Da die Forschungskonsortien, an denen bereits alle deutschen Universitätskliniken beteiligt sind, ihre Tätigkeit gerade erst aufgenommen haben, ergeben sich vielfältige Möglichkeiten, die hochwertigen Daten der Partnerstandorte für die Forschung in Zukunft zugänglich und nutzbar zu machen. Aufgrund des offenen Quellcodes von Kaapana, seiner auf gängigen Industriestandards basierenden Architektur und der bereits weiten Verbreitung könnte dieses Framework auch als Basis für die Datenverarbeitung in anderen Forschungsbereichen Anwendung finden. Somit könnte auch eine verbesserte Datenintegration für klinische Datenverarbeitung im Allgemeinen unterstützt werden.

Bibliography

[Aerts et al., 2019] Hugo J. W. L. Aerts, Leonard Wee, Emmanuel Rios Velazquez, Ralph T. H. Leijenaar, Chintan Parmar, Patrick Grossmann, Sara Carvalho, Johan Bussink, Ren Monshouwer, Benjamin Haibe-Kains, Derek Rietveld, Frank Hoebers, Michelle M. Rietbergen, C. Ren Leemans, Andre Dekker, John Quackenbush, Robert J. Gillies, and Philippe Lambin. **Data From NSCLC-Radiomics**, 2019.

[Akin et al., 2016] Oguz Akin, Pierre Elnajjar, Matthew Heller, Rose Jarosz, Bradley J. Erickson, Shanah Kirk, Yueh Lee, Marston W. Linehan, Rabindra Gautam, Raghu Vikram, Kimberly M. Garcia, Charles Roche, Ermelinda Bonaccio, and Joe Filippini. **Radiology Data from The Cancer Genome Atlas Kidney Renal Clear Cell Carcinoma [TCGA-KIRC] Collection**, 2016.

[Amazon, 2021a] Amazon. **Amazon Simple Storage Service S3**. https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html, November 2021a.

[Amazon, 2021b] Amazon. **Amazon Web Services (AWS) - Cloud Computing Services**. https://aws.amazon.com/, November 2021b.

[Andrei Varabyeu, 2021] Andrei Varabyeu. **ReportPortal**. https://reportportal.io/, November 2021.

[Andrey Fedorov, 2021] Andrey Fedorov. **QIICR**. http://qiicr.org/index.html, November 2021.

[Ansible, 2021] Ansible. **Ansible Is Simple IT Automation**. https://www.ansible.com, November 2021.

[Apache, 2021a] Apache. **Apache Airflow**. https://airflow.apache.org/, November 2021a.

[Apache, 2021b] Apache. **Apache Lucene Core**. https://lucene.apache.org/core/index.html, November 2021b.

[Apache, 2021c] Apache. **Apache Solr**. https://solr.apache.org/index.html, November 2021c.

[Bai et al., 2019] Junjie Bai, Fang Lu, Ke Zhang, et al. **ONNX: Open Neural Network Exchange**, 2019.

[Beichel et al., 2015] Reinhard R Beichel, Ethan J Ulrich, Christian Bauer, A Wahle, B Brown, T Chang, KA Plichta, BJ Smith, JJ Sunderland, T Braun, Andrey Fedorov, David Clunie, M Onken, Vincent A Magnotta, Yusuf Menda, J Riesmeier, Steve Pieper, Ron Kikinis, MM Graham, Thomas Casavant, M Sonka, and JM Buatti. **Data From QIN-HEADNECK**. *The Cancer Imaging Archive*, 2015. doi: 10.7937/K9/TCIA.2015.K0F5CGLI.

[Bercovich and Javitt, 2018] Eyal Bercovich and Marcia C. Javitt. **Medical Imaging: From Roentgen to the Digital Revolution, and Beyond**. *Rambam Maimonides Medical Journal*, 9(4):e0034, October 2018. ISSN 2076-9172. doi: 10.5041/RMMJ.10355.

[Brunnbauer, 2013] Michael Brunnbauer. **DICOM Metadata as RDF**. In Matthias Horbach, editor, *INFORMATIK 2013 – Informatik Angepasst an Mensch, Organisation Und Umwelt*, pages 1796–1804, Bonn, 2013. Gesellschaft für Informatik e.V.

[Chollet et al., 2015] Francois Chollet et al. **Keras**. https://github.com/fchollet/keras, 2015.

[CISPA, 2021] CISPA. **Trustworthy Federated Data Analytics (TFDA)**. https://tfda.hmsp.center/, November 2021.

[Cloud Native Computing Foundation, 2021] Cloud Native Computing Foundation. **Cri-o**. https://cri-o.io/, November 2021.

[CNCF, 2021] Serverless Working Group CNCF. **CNCF WG-Serverless Whitepaper v1.0**. https://github.com/cncf/wg-serverless, November 2021.

[Coder, 2021] Coder. **Code-Server GitHub**. Coder, September 2021.

[Committee, 2016] IHE Radiology Technical Committee. **IHE Radiology Technical Framework Supplement Invoke Image Display (IID)**, September 2016.

[Conquest, 2021] Conquest. **Conquest DICOM Software**. https://ingenium.home.xs4all.nl/dicom.html, November 2021.

[Darcy Mason, 2021] Darcy Mason. **Pydicom**. https://pydicom.github.io/, October 2021.

[Dash et al., 2019] Sabyasachi Dash, Sushil Kumar Shakyawar, Mohit Sharma, and Sandeep Kaushik. **Big Data in Healthcare: Management, Analysis and Future Prospects**. *Journal of Big Data*, 6(1):54, June 2019. ISSN 2196-1115. doi: 10.1186/s40537-019-0217-0.

[Deist et al., 2017] Timo Deist, Arthur Jochems, Johan van Soest, Georgi Nalbantov, Cary Oberije, Sean Walsh, Michael Eble, Paul Bulens, Philippe Coucke, Wim Dries, Andre Dekker, and Philippe Lambin. **Infrastructure and Distributed Learning Methodology for Privacy-Preserving Multi-Centric Rapid Learning Health Care: euroCAT**. *Clinical and Translational Radiation Oncology*, 4: 24–31, June 2017. ISSN 2405-6308. doi: 10.1016/j.ctro.2016.12.004.

[Deist et al., 2020] Timo M. Deist, Frank J. W. M. Dankers, Priyanka Ojha, M. Scott Marshall, Tomas Janssen, Corinne Faivre-Finn, Carlotta Masciocchi, Vincenzo Valentini, Jiazhou Wang, Jiayan Chen, Zhen Zhang, Emiliano Spezi, Mick Button, Joost Jan Nuyttens, René Vernhout, Johan van Soest, Arthur Jochems, René Monshouwer, Johan Bussink, Gareth Price, Philippe Lambin, and Andre Dekker. **Distributed Learning on 20 000+ Lung Cancer Patients – The Personal Health Train**. *Radiotherapy and Oncology*, 144:189–200, March 2020. ISSN 0167-8140. doi: 10.1016/j.radonc.2019.11.019.

[Denney et al., 2016] Michael J. Denney, Dustin M. Long, Matthew G. Armistead, Jamie L. Anderson, and Baqiyyah N. Conway. **Validating the Extract, Transform, Load Process Used to Populate a Large Clinical Research Database**. *International Journal of Medical Informatics*, 94:271–274, October 2016. ISSN 1386-5056. doi: 10.1016/j.ijmedinf.2016.07.009.

[Developers, 2021] TensorFlow Developers. **TensorFlow**. Zenodo, November 2021.

[DKTK, 2021] DKTK. **DKTK**. https://www.dktk.org/en/about-us/about-dktk, November 2021.

[Eichelberg et al., 2004] Marco Eichelberg, Joerg Riesmeier, Thomas Wilkens, Andrew J. Hewett, Andreas Barth, and Peter Jensch. **Ten Years of Medical Imaging Standardization and Prototypical Implementation: The DICOM Standard and the OFFIS DICOM Toolkit (DCMTK)**. In *Medical Imaging 2004: PACS and Imaging Informatics*, volume 5371, pages 57–68. SPIE, April 2004. doi: 10.1117/12.534853.

[Elastic, 2021] Elastic. **Elastic Stack: Elasticsearch, Kibana, Beats & Logstash**. https://www.elastic.co/elastic-stack, November 2021.

[Erickson et al., 2014] Bradley J. Erickson, Patricio Fajnwaks, Steve G. Langer, and John Perry. **Multisite Image Data Collection and Management Using the RSNA Image Sharing Network**. *Translational Oncology*, 7(1):36–39, February 2014. ISSN 1936-5233. doi: 10.1593/tlo.13799.

[Erickson et al., 2016] Bradley J. Erickson, Shanah Kirk, Yueh Lee, Oliver Bathe, Melissa Kearns, Cindy Gerdes, Kimberly Rieger-Christ, and John Lemmerman. **Radiology Data from The Cancer Genome Atlas Liver Hepatocellular Carcinoma [TCGA-LIHC] Collection**, 2016.

[Evan Bottcher, 2018] Evan Bottcher. **What I Talk About When I Talk About Platforms**. https://martinfowler.com/articles/talk-about-platforms.html, March 2018.

[Evan You, 2021] Evan You. **Vue.Js**. https://vuejs.org/, November 2021.

[Factor et al., 2005] M. Factor, K. Meth, D. Naor, O. Rodeh, and J. Satran. **Object Storage: The Future Building Block for Storage Systems**. In *2005 IEEE International Symposium on Mass Storage Systems and Technology*, pages 119–123, June 2005. doi: 10.1109/LGDI.2005.1612479.

[Falcon et al., 2020] William Falcon, Jirka Borovec, Adrian Wälchli, Nic Eggert, Justus Schock, Jeremy Jordan, Nicki Skafte, Ir1dXD, Vadim Bereznyuk, Ethan Harris, Tullie Murrell, Peter Yu, Sebastian Præsius, Travis Addair, Jacob Zhong, Dmitry Lipin, So Uchida, Shreyas Bapat, Hendrik Schröter, Boris Dayma, Alexey Karnachev, Akshay Kulkarni, Shunta Komatsu, Martin.B, Jean-Baptiste SCHIRATTI, Hadrien Mary, Donal Byrne, Cristobal Eyzaguirre, cinjon, and Anton Bakhtin. **PyTorchLightning**. Zenodo, May 2020.

[Fedorov et al., 2021] Andrey Fedorov, William J. R. Longabaugh, David Pot, David A. Clunie, Steve Pieper, Hugo J. W. L. Aerts, André Homeyer, Rob Lewis, Afshin Akbarzadeh, Dennis Bontempi, William Clifford, Markus D. Herrmann, Henning Höfener, Igor Octaviano, Chad Osborne, Suzanne Paquette, James Petts, Davide Punzo, Madelyn Reyes, Daniela P. Schacherer, Mi Tian, George White, Erik Ziegler, Ilya Shmulevich, Todd Pihl, Ulrike Wagner, Keyvan Farahani, and Ron Kikinis. **NCI Imaging Data Commons**. *Cancer Research*, 81(16):4188–4193, August 2021. ISSN 0008-5472, 1538-7445. doi: 10.1158/0008-5472.CAN-21-0950.

[Fedorov et al., 2016] Andriy Fedorov, David Clunie, Ethan Ulrich, Christian Bauer, Andreas Wahle, Bartley Brown, Michael Onken, Jörg Riesmeier, Steve Pieper, Ron Kikinis, John Buatti, and Reinhard R. Beichel. **DICOM for Quantitative**

Imaging Biomarker Development: A Standards Based Approach to Sharing Clinical Data and Structured PET/CT Analysis Results in Head and Neck Cancer Research. *PeerJ*, 4:e2057, May 2016. ISSN 2167-8359. doi: 10.7717/peerj.2057.

[Floca, 2014] Ralf Floca. **Challenges of Open Data in Medical Research**. In Sönke Bartling and Sascha Friesike, editors, *Opening Science*, pages 297–307. Springer International Publishing, Cham, 2014. ISBN 978-3-319-00025-1 978-3-319-00026-8. doi: 10.1007/978-3-319-00026-8_22.

[Gaye et al., 2014] Amadou Gaye, Yannick Marcon, Julia Isaeva, Philippe LaFlamme, Andrew Turner, Elinor M Jones, Joel Minion, Andrew W Boyd, Christopher J Newby, Marja-Liisa Nuotio, Rebecca Wilson, Oliver Butters, Barnaby Murtagh, Ipek Demir, Dany Doiron, Lisette Giepmans, Susan E Wallace, Isabelle Budin-Ljøsne, Carsten Oliver Schmidt, Paolo Boffetta, Mathieu Boniol, Maria Bota, Kim W Carter, Nick deKlerk, Chris Dibben, Richard W Francis, Tero Hiekkalinna, Kristian Hveem, Kirsti Kvaløy, Sean Millar, Ivan J Perry, Annette Peters, Catherine M Phillips, Frank Popham, Gillian Raab, Eva Reischl, Nuala Sheehan, Melanie Waldenberger, Markus Perola, Edwin van den Heuvel, John Macleod, Bartha M Knoppers, Ronald P Stolk, Isabel Fortier, Jennifer R Harris, Bruce HR Woffenbuttel, Madeleine J Murtagh, Vincent Ferretti, and Paul R Burton. **DataSHIELD: Taking the Analysis to the Data, Not the Data to the Analysis**. *International Journal of Epidemiology*, 43(6):1929–1944, December 2014. ISSN 1464-3685, 0300-5771. doi: 10.1093/ije/dyu188.

[Genereaux et al., 2018] Brad W. Genereaux, Donald K. Dennison, Kinson Ho, Robert Horn, Elliot Lewis Silver, Kevin O'Donnell, and Charles E. Kahn. **DICOMweb™: Background and Application of the Web Standard for Medical Imaging**. *Journal of Digital Imaging*, 31(3):321–326, June 2018. ISSN 1618-727X. doi: 10.1007/s10278-018-0073-z.

[Gibson et al., 2018] Eli Gibson, Wenqi Li, Carole Sudre, Lucas Fidon, Dzhoshkun I. Shakir, Guotai Wang, Zach Eaton-Rosen, Robert Gray, Tom Doel, Yipeng Hu, Tom Whyntie, Parashkev Nachev, Marc Modat, Dean C. Barratt, Sébastien Ourselin, M. Jorge Cardoso, and Tom Vercauteren. **NiftyNet: A Deep-Learning Platform for Medical Imaging**. *Computer Methods and Programs in Biomedicine*, 158: 113–122, May 2018. ISSN 1872-7565. doi: 10.1016/j.cmpb.2018.01.025.

[Gillies et al., 2015] Robert J. Gillies, Paul E. Kinahan, and Hedvig Hricak. **Radiomics: Images Are More than Pictures, They Are Data**. *Radiology*, 278(2):563–577, November 2015. ISSN 0033-8419. doi: 10.1148/radiol.2015151169.

[Goch et al., 2018] Caspar J. Goch, Jasmin Metzger, Martin Hettich, André Klein, Tobias Norajitra, Michael Götz, Jens Petersen, Klaus H. Maier-Hein, and Marco Nolden. **Automated Containerized Medical Image Processing Based on MITK and Python**. In Andreas Maier, Thomas M. Deserno, Heinz Handels, Klaus Hermann Maier-Hein, Christoph Palm, and Thomas Tolxdorff, editors, *Bildverarbeitung für die Medizin 2018*, Informatik aktuell, pages 315–315, Berlin, Heidelberg, 2018. Springer. ISBN 978-3-662-56537-7. doi: 10.1007/978-3-662-56537-7_82.

[Google, 2021] Google. **BigQuery: Cloud Data Warehouse**. https://cloud.google.com/bigquery, November 2021.

[Google, 2021] Google. **Cloud Computing Services**. https://cloud.google.com/, November 2021.

[Götz et al., 2019] Michael Götz, Marco Nolden, and Klaus Maier-Hein. **MITK Phenotyping: An Open-Source Toolchain for Image-Based Personalized Medicine with Radiomics**. *Radiotherapy and Oncology*, 131:108–111, February 2019. ISSN 01678140. doi: 10.1016/j.radonc.2018.11.021.

[Grafana Labs, 2021] Grafana Labs. **Grafana**. https://grafana.com/, November 2021.

[Grossberg et al., 2020] Aaron Grossberg, Hesham Elhalawani, Abdallah Mohamed, Sam Mulder, Bowman Williams, Aubrey L. White, James Zafereo, Andrew J. Wong, Joel E. Berends, Shady AboHashem, Jeremy M. Aymard, Aasheesh Kanwar, Subha Perni, Crosby D. Rock, Sasikarn Chamchod, Michael E. Kantor, Theodora Browne, Katherine A. Hutcheson, G. Brandon Gunn, Steven J. Frank, David Rosenthal, Adam S. Garden, Clifton Fuller, and M.D. Anderson Cancer Center Head and Neck Quantitative Imaging Working Group. **HNSCC**, 2020.

[Gruendner et al., 2019] Julian Gruendner, Thorsten Schwachhofer, Phillip Sippl, Nicolas Wolf, Marcel Erpenbeck, Christian Gulden, Lorenz A. Kapsner, Jakob Zierk, Sebastian Mate, Michael Stürzl, Roland Croner, Hans-Ulrich Prokosch, and Dennis Toddenroth. **KETOS: Clinical Decision Support and Machine Learning as a Service - A Training and Deployment Platform Based on Docker, OMOP-CDM, and FHIR Web Services**. *PloS One*, 14(10):e0223010, 2019. ISSN 1932-6203. doi: 10.1371/journal.pone.0223010.

[Gunter Zeilinger, 2021] Gunter Zeilinger. **Open Source Clinical Image and Object Management**. https://dcm4che.org/, October 2021.

[Harbor Authors, 2021] The Linux Foundation Harbor Authors. **Harbor**. https://goharbor.io/, November 2021.

[Healthineers, 2020] Siemens Healthineers. **AI-Rad Companion: Data Privacy and Security White Paper**, March 2020.

[Herrick et al., 2016] Rick Herrick, William Horton, Timothy Olsen, Michael McKay, Kevin A. Archie, and Daniel S. Marcus. **XNAT Central: Open Sourcing Imaging Research Data**. *NeuroImage*, 124:1093–1096, January 2016. ISSN 1053-8119. doi: 10.1016/j.neuroimage.2015.06.076.

[Herrmann et al., 2018] Markus D. Herrmann, David A. Clunie, Andriy Fedorov, Sean W. Doyle, Steven Pieper, Veronica Klepeis, Long P. Le, George L. Mutter, David S. Milstone, Thomas J. Schultz, Ron Kikinis, Gopal K. Kotecha, David H. Hwang, Katherine P. Andriole, A. John Iafrate, James A. Brink, Giles W. Boland, Keith J. Dreyer, Mark Michalski, Jeffrey A. Golden, David N. Louis, and Jochen K. Lennerz. **Implementing the DICOM Standard for Digital Pathology**. *Journal of Pathology Informatics*, 9(1):37, January 2018. ISSN 2153-3539. doi: 10.4103/jpi.jpi_42_18.

[HL7, 2019] HL7. **FHIR v4.0.1**, October 2019.

[Initiative, 2011] The Neuroimaging Informatics Technology Initiative. **NIfTI: — Neuroimaging Informatics Technology Initiative**. https://nifti.nimh.nih.gov/, March 2011.

[Isensee et al., 2020] Fabian Isensee, Paul Jäger, Jakob Wasserthal, David Zimmerer, Jens Petersen, Simon Kohl, Justus Schock, Andre Klein, Tobias Roß, Sebastian Wirkert, Peter Neher, Stefan Dinkelacker, Gregor Köhler, and Klaus Maier-Hein. **Batchgenerators - a Python Framework for Data Augmentation**. Zenodo, January 2020.

[Isensee et al., 2021] Fabian Isensee, Paul F. Jaeger, Simon A. A. Kohl, Jens Petersen, and Klaus H. Maier-Hein. **nnU-Net: A Self-Configuring Method for Deep Learning-Based Biomedical Image Segmentation**. *Nature Methods*, 18(2): 203–211, February 2021. ISSN 1548-7105. doi: 10.1038/s41592-020-01008-z.

[Jäger et al., 2017] Paul F. Jäger, Sebastian Bickelhaupt, Frederik Bernd Laun, Wolfgang Lederer, Daniel Heidi, Tristan Anselm Kuder, Daniel Paech, David Bonekamp, Alexander Radbruch, Stefan Delorme, Heinz-Peter Schlemmer, Franziska Steudle, and Klaus H. Maier-Hein. **Revealing Hidden Potentials of the Q-Space Signal in Breast Cancer**. In Maxime Descoteaux, Lena Maier-Hein, Alfred Franz, Pierre Jannin, D. Louis Collins, and Simon Duchesne, editors, *Medical Image Computing and Computer Assisted Intervention - MICCAI 2017*, Lecture Notes in Computer Science, pages 664–671,

Cham, 2017. Springer International Publishing. ISBN 978-3-319-66182-7. doi: 10.1007/978-3-319-66182-7_76.

[Jodogne, 2018] Sébastien Jodogne. **The Orthanc Ecosystem for Medical Imaging**. *Journal of Digital Imaging*, 31(3):341–352, June 2018. ISSN 1618-727X. doi: 10.1007/s10278-018-0082-y.

[Joel Martin, 2021] Joel Martin. **noVNC**. https://novnc.com/info.html, November 2021.

[Kata, 2021] Kata. **Kata Containers - Open Source Container Runtime Software**. https://katacontainers.io/, November 2021.

[Keycloak, 2021] Keycloak. **Keycloak**. https://www.keycloak.org/, November 2021.

[Kickingereder et al., 2019] Philipp Kickingereder, Fabian Isensee, Irada Tursunova, Jens Petersen, Ulf Neuberger, David Bonekamp, Gianluca Brugnara, Marianne Schell, Tobias Kessler, Martha Foltyn, Inga Harting, Felix Sahm, Marcel Prager, Martha Nowosielski, Antje Wick, Marco Nolden, Alexander Radbruch, Jürgen Debus, Heinz-Peter Schlemmer, Sabine Heiland, Michael Platten, Andreas von Deimling, Martin J. van den Bent, Thierry Gorlia, Wolfgang Wick, Martin Bendszus, and Klaus H. Maier-Hein. **Automated Quantitative Tumour Response Assessment of MRI in Neuro-Oncology with Artificial Neural Networks: A Multicentre, Retrospective Study**. *The Lancet Oncology*, 20(5):728–740, May 2019. ISSN 1470-2045, 1474-5488. doi: 10.1016/S1470-2045(19)30098-1.

[Kleesiek et al., 2020] Jens Kleesiek, Jacob M. Murray, Georgios Kaissis, and Rickmer Braren. **Künstliche Intelligenz und maschinelles Lernen in der onkologischen Bildgebung**. *Der Onkologe*, 26(1):60–65, January 2020. ISSN 1433-0415. doi: 10.1007/s00761-019-00679-4.

[Klein et al., 2020] Jan Klein, Markus Wenzel, Daniel Romberg, Alexander Köhn, Peter Kohlmann, Florian Link, Annika Hänsch, Volker Dicken, Ruben Stein, Julian Haase, Andreas Schreiber, Rainer Kasan, Horst Hahn, and Hans Meine. **QuantMed: Component-based Deep Learning Platform for Translational Research**. In Thomas M. Deserno and Po-Hao Chen, editors, *Medical Imaging 2020: Imaging Informatics for Healthcare, Research, and Applications*, page 28, Houston, United States, March 2020. SPIE. ISBN 978-1-5106-3403-9 978-1-5106-3404-6. doi: 10.1117/12.2549582.

[Kluyver et al., 2016] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. **Jupyter Notebooks – a Publishing Format for Reproducible**

Computational Workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90. IOS Press, 2016.

[Konstantinos Tsakalozos, 2021] Konstantinos Tsakalozos. **MicroK8s**. https://microk8s.io/, November 2021.

[kubernetes, 2021] kubernetes. **Kubernetes/Dashboard**. Kubernetes, July 2021.

[Kubernetes, 2021] Kubernetes. **Kubernetes Python Client**. Kubernetes Clients, August 2021.

[Lablans et al., 2015] Martin Lablans, Andreas Borg, and Frank Ückert. **A RESTful Interface to Pseudonymization Services in Modern Web Applications**. *BMC Medical Informatics and Decision Making*, 15(1):2, February 2015. ISSN 1472-6947. doi: 10.1186/s12911-014-0123-5.

[Lasso, 2019] Andras Lasso. **AI-assisted Segmentation Extension - Announcements - 3D Slicer Community**. https://discourse.slicer.org/t/ai-assisted-segmentation-extension/9536, December 2019.

[louketo, 2021] louketo. **Louketo-Proxy**. Louketo, July 2021.

[Ludovic Fernandez, 2021] Ludovic Fernandez. **Traefik**. https://doc.traefik.io/traefik/, November 2021.

[Maier-Hein et al., 2018] Lena Maier-Hein, Matthias Eisenmann, Annika Reinke, Sinan Onogur, Marko Stankovic, Patrick Scholz, Tal Arbel, Hrvoje Bogunovic, Andrew P. Bradley, Aaron Carass, Carolin Feldmann, Alejandro F. Frangi, Peter M. Full, Bram van Ginneken, Allan Hanbury, Katrin Honauer, Michal Kozubek, Bennett A. Landman, Keno März, Oskar Maier, Klaus Maier-Hein, Bjoern H. Menze, Henning Müller, Peter F. Neher, Wiro Niessen, Nasir Rajpoot, Gregory C. Sharp, Korsuk Sirinukunwattana, Stefanie Speidel, Christian Stock, Danail Stoyanov, Abdel Aziz Taha, Fons van der Sommen, Ching-Wei Wang, Marc-André Weber, Guoyan Zheng, Pierre Jannin, and Annette Kopp-Schneider. **Why Rankings of Biomedical Image Analysis Competitions Should Be Interpreted with Care**. *Nature Communications*, 9(1):5217, December 2018. ISSN 2041-1723. doi: 10.1038/s41467-018-07619-7.

[MATLAB, 2010] MATLAB. **Version 7.10.0 (R2010a)**. The MathWorks Inc., Natick, Massachusetts, 2010.

[Mell and Grance, 2011] Peter Mell and Tim Grance. **The NIST Definition of Cloud Computing**. Technical Report NIST Special Publication (SP) 800-145, National Institute of Standards and Technology, September 2011.

[Merkel, 2014] Dirk Merkel. **Docker: Lightweight Linux Containers for Consistent Development and Deployment**. *Linux journal*, 2014(239):2, 2014.

[Microsoft, 2021] Microsoft. **Cloud Computing Services | Microsoft Azure**. https://azure.microsoft.com/en-us/, November 2021.

[Microsoft, 2021] Microsoft. **Visual Studio Code**. https://code.visualstudio.com/, November 2021.

[MinIO, 2021] MinIO. **MinIO**. https://min.io, November 2021.

[MITK, 2019] MITK. **NvidiaAnnotationPlugin - Mitk.Org**. https://www.mitk.org/wiki/NvidiaAnnotationPlugin, March 2019.

[MONAI Deploy Working Group, 2021] MONAI Deploy Working Group. **MONAI Deploy Working Group**. https://github.com/Project-MONAI/monai-deploy, November 2021.

[MONAI initiative, 2021] MONAI initiative. **MONAI**. https://monai.io/, November 2021.

[Mukherjee, 2017] Siddhartha Mukherjee. **A.I. Versus M.D.** *The New Yorker*, March 2017. ISSN 0028-792X.

[NEMA, 2021a] NEMA. **Attribute Confidentiality Profiles**. http://dicom.nema.org/dicom/2013/output/chtml/part15/chapter_E.html, November 2021a.

[NEMA, 2021b] NEMA. **DICOM - DIMSE-C**. http://dicom.nema.org/dicom/2013/output/chtml/part07 November 2021b.

[NEMA, 2021c] NEMA. **DICOM JSON Model**. http://dicom.nema.org/dicom/2013/output/chtml/part November 2021c.

[NEMA, 2021a] Medical Imaging & Technology Alliance NEMA. **DICOM - C.8.20 Segmentation**. http://dicom.nema.org/medical/Dicom/2018d/output/chtml/part03/sect_C.8.20.htm November 2021a.

[NEMA, 2021b] Medical Imaging & Technology Alliance NEMA. **DICOM Information Model**. https://dicom.nema.org/medical/dicom/current/output/chtml/part04/chapter_6.html, November 2021b.

[NEMA, 2021c] Medical Imaging & Technology Alliance NEMA. **Encapsulated Document IOD**. http://dicom.nema.org/dicom/2013/output/chtml/part03/sect_A.45.html, November 2021c.

[NEMA, 2021d] Medical Imaging & Technology Alliance NEMA. **NEMA PS3 / ISO 12052, Digital Imaging and Communications in Medicine (DICOM) Standard**, 2021d.

[NEMA, 2021e] Medical Imaging & Technology Alliance NEMA. **Structured Report Document Information Object Definitions**. http://dicom.nema.org/dicom/2013/output/chtml/part03/sect_A.35.html, November 2021e.

[Newitt and Hylton, 2016] David Newitt and Nola Hylton. **Single Site Breast DCE-MRI Data and Segmentations from Patients Undergoing Neoadjuvant Chemotherapy**, 2016.

[nipy.org, 2021] nipy.org. **Neuroimaging in Python — NiBabel**. https://nipy.org/nibabel/, November 2021.

[Norajitra and Maier-Hein, 2017] T. Norajitra and K. H. Maier-Hein. **3D Statistical Shape Models Incorporating Landmark-Wise Random Regression Forests for Omni-Directional Landmark Detection**. *IEEE Transactions on Medical Imaging*, 36(1):155–168, January 2017. ISSN 0278-0062. doi: 10.1109/TMI.2016.2600502.

[NUM, 2021] NUM. **RACOON | Netzwerk Universitätsmedizin**. https://www.netzwerk-universitaetsmedizin.de/projekte/racoon, November 2021.

[NVIDIA, 2018] NVIDIA. **NVIDIA Clara Imaging**. https://developer.nvidia.com/clara-medical-imaging, September 2018.

[NVIDIA, 2019] NVIDIA. **NVIDIA Clara**. https://developer.nvidia.com/clara, May 2019.

[NVIDIA, 2020] NVIDIA. **Jumpstarting AI with a COVID-19 CT Inference Pipeline and the NVIDIA Clara Deploy QuickStart Virtual Machine**. https://developer.nvidia.com/blog/jumpstarting-ai-with-covid-19-ct-inference-pipeline-and-clara-deploy-quickstart-vm/, December 2020.

[NVIDIA NGC, 2021] NVIDIA NGC. **PyTorch | NVIDIA NGC**. https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pytorch, November 2021.

[OCI, 2021] The Linux Foundation OCI. **The Open Container Initiative**. https://opencontainers.org/about/overview/, November 2021.

[OFFIS, 2021a] OFFIS. **DCMTK - DICOM Toolkit**. https://dicom.offis.de/dcmtk.php.en, January 2021a.

[OFFIS, 2021b] OFFIS. **DCMTK: Dcm2json: Convert DICOM File and Data Set to JSON**. https://support.dcmtk.org/docs/dcm2json.html, January 2021b.

[OFFIS, 2021c] OFFIS. **DCMTK: Dcmsend: Simple DICOM Storage SCU (Sender)**. https://support.dcmtk.org/docs/dcmsend.html, January 2021c.

[OHDSI, 2021] OHDSI. **The Book of OHDSI - Chapter 4: The Common Data Model**. Observational Health Data Sciences and Informatics, January 2021.

[OpenClinica, 2021] OpenClinica. **OpenClinica | Clinical Data Management, EPRO and eCRF**. https://www.openclinica.com/, November 2021.

[Paszke et al., 2019] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. **PyTorch: An Imperative Style, High-Performance Deep Learning Library**. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[Perrey and Lycett, 2003] R. Perrey and M. Lycett. **Service-Oriented Architecture**. In *2003 Symposium on Applications and the Internet Workshops, 2003. Proceedings.*, pages 116–119, January 2003. doi: 10.1109/SAINTW.2003.1210138.

[Petersen et al., 2019] Jens Petersen, Paul F. Jäger, Fabian Isensee, Simon A. A. Kohl, Ulf Neuberger, Wolfgang Wick, Jürgen Debus, Sabine Heiland, Martin Bendszus, Philipp Kickingereder, and Klaus H. Maier-Hein. **Deep Probabilistic Modeling of Glioma Growth**. In Dinggang Shen, Tianming Liu, Terry M. Peters, Lawrence H. Staib, Caroline Essert, Sean Zhou, Pew-Thian Yap, and Ali Khan, editors, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*, Lecture Notes in Computer Science, pages 806–814, Cham, 2019. Springer International Publishing. ISBN 978-3-030-32245-8. doi: 10.1007/978-3-030-32245-8_89.

[Pezoa et al., 2016] Felipe Pezoa, Juan L Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. **Foundations of JSON Schema**. In *Proceedings of the 25th International Conference on World Wide Web*, pages 263–273. International World Wide Web Conferences Steering Committee, 2016.

[Philips, 2019] Philips. **Philips IntelliSpace Discovery 3 - End-to-end AI Solution for Medical Research**, 2019.

[Prometheus, 2021] Prometheus. **Prometheus - Monitoring System & Time Series Database**. https://prometheus.io/, November 2021.

[Quantitative Image Informatics for Cancer Research, 2021] Quantitative Image Informatics for Cancer Research. **Dcmqi Github Repository**. Quantitative Image Informatics for Cancer Research, July 2021.

[Rascovsky et al., 2012] Simón J. Rascovsky, Jorge A. Delgado, Alexander Sanz, Víctor D. Calvo, and Gabriel Castrillón. **Informatics in Radiology: Use of CouchDB for Document-based Storage of DICOM Objects**. *RadioGraphics*, 32(3):913–927, May 2012. ISSN 0271-5333. doi: 10.1148/rg.323115049.

[Ravindra and Grama, 2021] Vikram Ravindra and Ananth Grama. **De-Anonymization Attacks on Neuroimaging Datasets**. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD/PODS '21, pages 2394–2398, New York, NY, USA, June 2021. Association for Computing Machinery. ISBN 978-1-4503-8343-1. doi: 10.1145/3448016.3457234.

[Rensin, 2015] David K. Rensin. **Kubernetes - Scheduling the Future at Cloud Scale**. In *OSCON 2015*, page All. O'Reilly Media, Inc., 1005 Gravenstein Highway North Sebastopol, CA 95472, 2015.

[Rocher et al., 2019] Luc Rocher, Julien M. Hendrickx, and Yves-Alexandre de Montjoye. **Estimating the Success of Re-Identifications in Incomplete Datasets Using Generative Models**. *Nature Communications*, 10(1):3069, July 2019. ISSN 2041-1723. doi: 10.1038/s41467-019-10933-3.

[RSNA MIRC, 2021] RSNA MIRC. **MIRC CTP**. http://mircwiki.rsna.org/index.php?title=CTP-The_RSNA_Clinical_Trial_Processor, November 2021.

[Rubin et al., 2019] Daniel L. Rubin, Mete Ugur Akdogan, Cavit Altindag, and Emel Alkim. **ePAD: An Image Annotation and Analysis Platform for Quantitative Imaging**. *Tomography (Ann Arbor, Mich.)*, 5(1):170–183, March 2019. ISSN 2379-139X. doi: 10.18383/j.tom.2018.00055.

[Saltz et al., 2021] Joel Saltz, Mary Saltz, Prateek Prasanna, Richard Moffitt, Janos Hajagos, Erich Bremer, Joseph Balsamo, and Tahsin Kurc. **Stony Brook University COVID-19 Positive Cases**, 2021.

[Scherer et al., 2020a] Jonas Scherer, Klaus Kades, Hanno Gao, Ralf Floca, Peter Neher, Marco Nolden, and Klaus Maier-Hein. **Kaapana/Kaapana: V0.1.1**. Zenodo, October 2020a.

[Scherer et al., 2020b] Jonas Scherer, Marco Nolden, Jens Kleesiek, Jasmin Metzger, Klaus Kades, Verena Schneider, Michael Bach, Oliver Sedlaczek, Andreas M. Bucher, Thomas J. Vogl, Frank Grünwald, Jens-Peter Kühn, Ralf-Thorsten Hoffmann, Jörg Kotzerke, Oliver Bethge, Lars Schimmöller, Gerald Antoch, Hans-Wilhelm Müller, Andreas Daul, Konstantin Nikolaou, Christian la Fougère, Wolfgang G. Kunz, Michael Ingrisch, Balthasar Schachtner, Jens Ricke, Peter Bartenstein, Felix Nensa, Alexander Radbruch, Lale Umutlu, Michael Forsting, Robert Seifert, Ken Herrmann, Philipp Mayer, Hans-Ulrich Kauczor, Tobias Penzkofer, Bernd Hamm, Winfried Brenner, Roman Kloeckner, Christoph Düber, Mathias Schreckenberger, Rickmer Braren, Georgios Kaissis, Marcus Makowski, Matthias Eiber, Andrei Gafita, Rupert Trager, Wolfgang A. Weber, Jakob Neubauer, Marco Reisert, Michael Bock, Fabian Bamberg, Jürgen Hennig, Philipp Tobias Meyer, Juri Ruf, Uwe Haberkorn, Stefan O. Schoenberg, Tristan Kuder, Peter Neher, Ralf Floca, Heinz-Peter Schlemmer, and Klaus Maier-Hein. **Joint Imaging Platform for Federated Clinical Data Analytics**. *JCO Clinical Cancer Informatics*, 4:1027–1038, November 2020b. doi: 10.1200/CCI.20.00045.

[Shi et al., 2019] Zhenwei Shi, Ivan Zhovannik, Alberto Traverso, Frank J. W. M. Dankers, Timo M. Deist, Petros Kalendralis, René Monshouwer, Johan Bussink, Rianne Fijten, Hugo J. W. L. Aerts, Andre Dekker, and Leonard Wee. **Distributed Radiomics as a Signature Validation Study Using the Personal Health Train Infrastructure**. *Scientific Data*, 6(1):218, October 2019. ISSN 2052-4463. doi: 10.1038/s41597-019-0241-0.

[Skripcak et al., 2016] Tomas Skripcak, Uwe Just, Monique Simon, Daniel Buttner, Armin Luhr, Michael Baumann, and Mechthild Krause. **Toward Distributed Conduction of Large-Scale Studies in Radiation Therapy and Oncology: Open-Source System Integration Approach**. *IEEE Journal of Biomedical and Health Informatics*, 20(5):1397–1403, September 2016. ISSN 2168-2194, 2168-2208. doi: 10.1109/JBHI.2015.2450833.

[teem, 2021] teem. **Nrrd**. http://teem.sourceforge.net/nrrd/index.html, November 2021.

[TensorFlow, 2021] TensorFlow. **TensorBoard**. https://www.tensorflow.org/tensorboard, November 2021.

[The Linux Foundation, 2021] The Linux Foundation. **Helm**. https://helm.sh/, November 2021.

[The OpenStack Foundation, 2021] The OpenStack Foundation. **Open Source Cloud Computing Infrastructure**. https://www.openstack.org/, November 2021.

[Urban et al., 2017] Trinity Urban, Erik Ziegler, Rob Lewis, Chris Hafey, Cheryl Sadow, Annick D. Van den Abbeele, and Gordon J. Harris. **LesionTracker: Extensible Open-Source Zero-Footprint Web Viewer for Cancer Imaging Research and Clinical Trials**. *Cancer Research*, 77(21):e119–e122, November 2017. ISSN 0008-5472, 1538-7445. doi: 10.1158/0008-5472.CAN-17-0334.

[van Griethuysen et al., 2017] Joost J. M. van Griethuysen, Andriy Fedorov, Chintan Parmar, Ahmed Hosny, Nicole Aucoin, Vivek Narayan, Regina G. H. Beets-Tan, Jean-Christophe Fillion-Robin, Steve Pieper, and Hugo J. W. L. Aerts. **Computational Radiomics System to Decode the Radiographic Phenotype**. *Cancer Research*, 77(21):e104–e107, November 2017. ISSN 0008-5472, 1538-7445. doi: 10.1158/0008-5472.CAN-17-0339.

[VAN SOEST et al., 2014] Johan VAN SOEST, Tim LUSTBERG, Detlef GRITTNER, M. Scott MARSHALL, Lucas PERSOON, Bas NIJSTEN, Peter FELTENS, and Andre DEKKER. **Towards a Semantic PACS: Using Semantic Web Technology to Represent Imaging Data**. *Studies in health technology and informatics*, 205:166–170, 2014. ISSN 0926-9630.

[van Soest et al., 2018] Johan van Soest, Chang Sun, Ole Mussmann, Marco Puts, Bob van den Berg, Alexander Malic, Claudia van Oppen, David Townend, Andre Dekker, and Michel Dumontier. **Using the Personal Health Train for Automated and Privacy-Preserving Analytics on Vertically Partitioned Data**. *Studies in Health Technology and Informatics*, 247:581–585, 2018. ISSN 1879-8365.

[Wilkinson et al., 2016] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J. G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A. C. 't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. **The FAIR Guiding Principles for Scientific Data Management and Stewardship**. *Scientific Data*, 3(1):160018, March 2016. ISSN 2052-4463. doi: 10.1038/sdata.2016.18.

[Wolf et al., 2005] Ivo Wolf, Marcus Vetter, Ingmar Wegner, Thomas Böttger, Marco Nolden, Max Schöbinger, Mark Hastenteufel, Tobias Kunert, and Hans-Peter

Meinzer. **The Medical Imaging Interaction Toolkit**. *Medical Image Analysis*, 9(6):594–604, December 2005. ISSN 1361-8415. doi: 10.1016/j.media.2005.04.005.

[Yegnanarayana, 2009] Bayya Yegnanarayana. **Artificial Neural Networks**. PHI Learning Pvt. Ltd., 2009.

[Ziegler et al., 2020] Erik Ziegler, Trinity Urban, Danny Brown, James Petts, Steve D. Pieper, Rob Lewis, Chris Hafey, and Gordon J. Harris. **Open Health Imaging Foundation Viewer: An Extensible Open-Source Framework for Building Web-Based Imaging Applications to Support Cancer Research**. *JCO Clinical Cancer Informatics*, 4:336–345, November 2020. doi: 10.1200/CCI.19.00131.

[Zwanenburg et al., 2020] Alex Zwanenburg, Martin Vallières, Mahmoud A. Abdalah, Hugo J. W. L. Aerts, Vincent Andrearczyk, Aditya Apte, Saeed Ashrafinia, Spyridon Bakas, Roelof J. Beukinga, Ronald Boellaard, Marta Bogowicz, Luca Boldrini, Irène Buvat, Gary J. R. Cook, Christos Davatzikos, Adrien Depeursinge, Marie-Charlotte Desseroit, Nicola Dinapoli, Cuong Viet Dinh, Sebastian Echegaray, Issam El Naqa, Andriy Y. Fedorov, Roberto Gatta, Robert J. Gillies, Vicky Goh, Michael Götz, Matthias Guckenberger, Sung Min Ha, Mathieu Hatt, Fabian Isensee, Philippe Lambin, Stefan Leger, Ralph T.H. Leijenaar, Jacopo Lenkowicz, Fiona Lippert, Are Losnegård, Klaus H. Maier-Hein, Olivier Morin, Henning Müller, Sandy Napel, Christophe Nioche, Fanny Orlhac, Sarthak Pati, Elisabeth A.G. Pfaehler, Arman Rahmim, Arvind U.K. Rao, Jonas Scherer, Muhammad Musib Siddique, Nanna M. Sijtsema, Jairo Socarras Fernandez, Emiliano Spezi, Roel J.H.M. Steenbakkers, Stephanie Tanadini-Lang, Daniela Thorwarth, Esther G.C. Troost, Taman Upadhaya, Vincenzo Valentini, Lisanne V. van Dijk, Joost van Griethuysen, Floris H.P. van Velden, Philip Whybra, Christian Richter, and Steffen Löck. **The Image Biomarker Standardization Initiative: Standardized Quantitative Radiomics for High-Throughput Image-based Phenotyping**. *Radiology*, 295(2):328–338, May 2020. ISSN 0033-8419. doi: 10.1148/radiol.2020191145.

Own Contributions

The purpose of this chapter is to give an overview of my contributions and to differentiate these from whole team efforts. This thesis was written in the division of Medical Image Computing (MIC) headed by Prof. Dr. Klaus Maier-Hein, as part of a multi-disciplinary team of scientists. Prof. Dr. Klaus Maier-Hein was also the primary supervisor for this thesis. Throughout the entire time of my thesis, I was closely collaborating with members of Dr. Klaus Maier-Hein's group and partners from the consortia involved.

Own Contribution
In general, I was responsible for the design of all software concepts presented in this document and was also involved in all implementations described, however, I was not the main developer of all components. As the Kaapana framework has evolved into a larger software project, some of the components described in this thesis have been implemented by colleagues. For this reason, I will disclose in this section who contributed components relevant to this work that were not developed by me.

Share of Others
For the implementation of the framework core, this applies to the integration of desktop software components described in subsection 3.3.2. The noVNC base-container that enables such integrations was mainly developed and integrated by Tobias Stein and Hanno Gao. Likewise, the user interface described in subsection 3.3.3, including the extensions and the associated website development, was contributed by Klaus Kades.

Furthermore, some Case Scenarios were also implemented and integrated by others, since such integration by a third party was part of the concept to be evaluated. The design and implementation of this body part prediction described in Case Scenario 4.3.1 has been developed in the context of a master thesis by Sarah Schuhegger, which

has been partly supervised by me. Case Scenario 4.3.2: "Evaluation of Third-Party Workflow Development" of the second use case was accomplished by Erik Prescher from the University Hospital Frankfurt. qPSMA, which was evaluated for Case Scenario 4.3.3: On-Demand Services and Interactive Processing-Pipelines, was also integrated into the infrastructure by Klaus Kades. Finally, the customized MITK flavor needed for Use Case 4: "Interactive Data Annotation" was provided by Hanno Gao.

Since Kaapana has been adopted as the basis for several projects, many adjustments and bug fixes have been committed by colleagues in the meantime. The core development team currently consists of about 8 people and is constantly growing.

Own Publications

In this section, all publications that I was a part of and that contributed to my Ph.D. work are listed. It is subdivided into *First Authorships*, *Co-Authorships* and Software.

First Authorships - Peer Reviewed Journal Publications

Scherer, J, Nolden, M., Kleesiek, J., Metzger, J., Kades, K., Schneider, V., Bach, M., Sedlaczek, O., Bucher, A. M., Vogl, T. J., Grünwald, F., Kühn, J.-P., Hoffmann, R.-T., Kotzerke, J., Bethge, O., Schimmöller, L., Antoch, G., Müller, H.-W., Daul, A., ... Maier-Hein, K. (2020). Joint Imaging Platform for Federated Clinical Data Analytics. JCO Clinical Cancer Informatics, 4, 1027–1038. https://doi.org/10.1200/cci.20.00045

First Authorships - Peer Reviewed Conferences

Scherer, J., Nolden, M., Kleesiek, J., Metzger, J., Kades, K., Schneider, V., Gao, H., Neher, P., Floca, R., Schlemmer, H.-P., & Maier-Hein, K. (2021). Abstract: Joint Imaging Platform for Federated Clinical Data Analytics. In Bildverarbeitung für die Medizin 2021 (pp. 127–127). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-33198-6_31

Scherer, J., Kleesiek, J., Nolden, M., Kades, K., Schneider, V., Metzger, J., Maier-Hein, K., & Schlemmer, H. (2020, April). Die Joint Imaging Platform (JIP) des Deutschen Konsortiums für Translationale Krebsforschung (DKTK). 101. Deutscher Röntgenkongress Und 9. Gemeinsamer Kongress Der DRG Und ÖRG. 101. Deutscher Röntgenkongress und 9. Gemeinsamer Kongress der DRG und ÖRG. https://doi.org/10.1055/s-0040-1703123

Co-Authorships

Stein, T., Metzger, **J., Scherer**, J., Isensee, F., Norajitra, T., Kleesiek, J., Maier-Hein, K., & Nolden, M. (2019). Efficient Web-Based Review for Automatic Segmentation of Volumetric DICOM Images. In Informatik aktuell (pp. 158–163). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-25326-4_33

Software

Kaapana v0.1.1

Scherer J., Kades, K., Gao, H., Floca R., Neher P., Nolden M., Maier-Hein K.
DOI: `https://doi.org/10.5281/zenodo.5786866`
URL: `https://github.com/kaapana/kaapana`

Acknowledgements

Working on this dissertation over the past few years has been an intense and enriching experience for me. I would like to express my sincere gratitude to all those who have supported, guided and accompanied me along the way.

Special thanks go to my supervisor Prof Klaus Maier-Hein for giving me the opportunity to work on this thesis and for the great support whenever it was needed. I want to thank the members of my thesis advisory committee, Prof. Heinz-Peter Schlemmer and Dr. Urs Eisenmann, for their support and mentoring during this endeavor and Dr. Peter Neher, Dr. Marco Nolden and Dr. Ralf Floca for their great mentoring, guidance and daily collaboration on the various projects.

I am very grateful to my close teammate Klaus Kades, with whom I was able to master even the most difficult and challenging situations and who always jumped in when there was a need. I also had great pleasure working with Prof Jens Kleesiek, Jasmin Metzger and Verena Schneider on the JIP project, which was successful despite the great challenge thanks to the extraordinary team. Likewise, I would like to express my acknowledgement to the steadily growing Kaapana team for the good collaborative work on the technical basis.

Close collaboration with hospitals and affiliated scientists is essential for successful research on such a distributed infrastructure. Many thanks to Dr. Tobias Penzkofer and Dr. Andreas Bucher for their continuous interest and commitment, whom I would like to mention here on behalf of the many cooperation partners of the JIP and RACOON consortia.

Finally, I would like to express my deepest gratitude to my friends and especially to my family. The endless conversations and plenty of advice have made this work possible in the first place. From the bottom of my heart, thank you.

Eidesstattliche Versicherung

1. Bei der eingereichten Dissertation zu dem Thema "***Decentralized Infrastructure for Medical Image Analysis***" handelt es sich um meine eigenständig erbrachte Leistung.

2. Ich habe nur die angegebenen Quellen und Hilfsmittel benutzt und mich keiner unzulässigen Hilfe Dritter bedient. Insbesondere habe ich wörtlich oder sinngemäß aus anderen Werken übernommene Inhalte als solche kenntlich gemacht.

3. Die Richtigkeit der vorstehenden Erklärungen bestätige ich.

4. Die Bedeutung der eidesstattlichen Versicherung und die strafrechtlichen Folgen einer unrichtigen oder unvollständigen eidesstattlichen Versicherung sind mir bekannt. Ich versichere an Eides statt, dass ich nach bestem Wissen die reine Wahrheit erklärt und nichts verschwiegen habe.

Heidelberg, 11.01.2022

_____

Jonas Scherer