INAUGURAL-DISSERTATION

submitted to the

Combined Faculty of Mathematics, Engineering and Natural Sciences

of the

Ruprecht–Karls–University

Heidelberg

for the degree of

Doctor of Natural Sciences

put forward by

Bernhard Klein, M.Sc.

born in

Böblingen, Baden-Württemberg

Date of oral exam: 16.12.2025

Resource-Efficient and Robust Inference of Deep and Bayesian Neural Networks on Embedded and Analog Computing Platforms

A Study of Compression, Robustness, and Bayesian Inference from Embedded Processors to Analog Photonic Accelerators

Advisor: Prof. Dr. Holger Fröning

Heidelberg, 2025

To my family and friends,
and especially to my loving parents and sister,
for their unwavering support and belief in me.

Abstract

While modern machine learning has transformed numerous application domains, its growing computational demands increasingly constrain scalability and efficiency, particularly on embedded and resource-constrained platforms. In practical deployments, neural networks must not only operate efficiently but also provide reliable predictions when faced with distributional changes or previously unseen data. Bayesian neural networks offer a principled framework for quantifying uncertainty, but their higher computational demands further compound these challenges.

This work advances *resource-efficient and robust inference* for both conventional and Bayesian neural networks through the joint pursuit of algorithmic and hardware efficiency. The former reduces computational cost through model compression and approximate Bayesian inference, while the latter optimizes mapping to digital accelerators and explores novel analog hardware platforms, bridging algorithmic optimization and physical realization.

The first contribution introduces the *Galen* framework, which performs automatic, layer-specific compression guided by sensitivity analysis and hardware-in-the-loop feedback, jointly optimizing quantization and pruning to balance accuracy and efficiency on embedded devices. As analog accelerators offer additional efficiency gains at the cost of noise, their modeling exposes device imperfections, while a layer-wise analysis reveals how networks learn to tolerate such effects during training. This work extends noisy training to nonstationary conditions, thereby enhancing robustness and stability in analog hardware.

A complementary line of work advances probabilistic inference. Building on insights into Bayesian-neural-network design and training, this work develops analytic and ensemble-based approximations that replace costly sampling, integrates them into a compiler stack, and optimizes them for probabilistic inference on embedded hardware. Finally, *probabilistic photonic computing* introduces a novel paradigm in which controlled analog noise serves as an intrinsic entropy source, enabling ultrafast and energy-efficient probabilistic inference directly in hardware.

Together, these studies demonstrate how *efficiency and reliability can be advanced jointly* through the co-design of algorithms, compilers, and hardware, laying the foundation for the next generation of *trustworthy and energy-efficient* machine-learning systems.

Zusammenfassung

Moderne Verfahren des maschinellen Lernens haben zahlreiche Anwendungsfelder grundlegend verändert. Mit dem stetig wachsenden Rechenbedarf stoßen sie jedoch zunehmend an Grenzen hinsichtlich Skalierbarkeit und Effizienz – insbesondere auf eingebetteten und ressourcenbeschränkten Plattformen. Bei der Anwendung in realen Systemen müssen neuronale Netze nicht nur effizient arbeiten, sondern auch unter sich verändernden Datenverteilungen oder bei bislang unbekannten Datenpunkten verlässliche Vorhersagen liefern. Bayessche neuronale Netze bieten hierfür einen konsistenten theoretischen Rahmen zur Quantifizierung von Unsicherheiten, ihr zusätzlicher Rechenaufwand verstärkt diese Herausforderungen jedoch weiter.

Diese Arbeit verfolgt das Ziel einer *ressourceneffizienten und robusten Inferenz* sowohl für konventionelle als auch für bayessche neuronale Netze durch die gemeinsame Optimierung von Algorithmen und Hardware. Die algorithmische Effizienz wird durch Modellkompression und approximative bayessche Verfahren verbessert, während die Hardwareeffizienz sowohl die Abbildung auf digitale Beschleuniger als auch die Erforschung neuartiger analoger Plattformen umfasst und damit eine Brücke zwischen algorithmischer Optimierung und hardwareseitiger Realisierung schlägt.

Den ersten Beitrag stellt das *Galen*-Framework dar, das eine automatische, feinaufgelöste Kompression auf Grundlage von Sensitivitätsanalysen und Hardware-in-the-Loop-Rückkopplung durchführt. Quantisierung und Pruning werden dabei gemeinsam optimiert, um Genauigkeit und Effizienz auf eingebetteten Systemen in Einklang zu bringen. Da analoge Beschleuniger zusätzliche Effizienzgewinne auf Kosten von Rechenrauschen bieten, werden ihre Nichtidealitäten modelliert. Eine Analyse auf Ebene der Netzwerkschichten zeigt, wie neuronale Netze lernen, solche Störungen während des Trainings zu tolerieren. Darauf aufbauend erweitert diese Arbeit das Training mit Rauschinjektion auf nichtstationäre Bedingungen, wodurch Robustheit und Stabilität in analogen Beschleunigern gesteigert werden.

Ein weiterer Schwerpunkt liegt auf der probabilistischen Inferenz. Aufbauend auf Erkenntnissen zum Entwurf und Training bayesscher neuronaler Netze werden effiziente analytische und ensemblebasierte Approximationen entwickelt, die aufwändiges Sampling ersetzen und in einer Compiler-Infrastruktur mit optimierten probabilistischen Operatoren für eingebettete Hardware umgesetzt sind. Schließlich wird mit dem *probabilistischen photonischen Rechnen* ein neuartiges

Paradigma eingeführt, bei dem kontrolliertes analoges Rauschen als intrinsische Entropiequelle dient und ultraschnelle, energieeffiziente probabilistische Inferenz direkt in photonischer Hardware ermöglicht.

Zusammenfassend zeigt diese Arbeit, dass *Effizienz und Zuverlässigkeit gemeinsam gesteigert werden können*, wenn Algorithmen, Compiler und Hardware als integriertes System konzipiert werden. Damit wird das Fundament für die nächste Generation vertrauenswürdiger und energieeffizienter Systeme des maschinellen Lernens gelegt.

Contents

1

Introduction

The extraordinary progress of machine learning (ML) over the past decade has been driven by increasingly large models, massive datasets, and powerful compute infrastructures. While these advances have led to remarkable performance across domains such as computer vision [214], speech recognition [205], and natural language processing [85], they come at the cost of substantial computational and energy demands. The growing size and complexity of modern deep neural networks (DNNs) [34], [195], together with the end of *Dennard scaling*—the breakdown of power efficiency gains from transistor miniaturization [218]—have made throughput and energy efficiency critical bottlenecks for both large-scale training [47] and on-device inference [150], [178]. These efficiency constraints have intensified the need for methods that preserve predictive performance under limited computational and energy budgets, driving progress in both algorithmic optimization and hardware specialization.

At the same time, DNNs, despite their empirical success, fundamentally lack a principled mechanism to quantify uncertainty. In classification tasks, for example, the softmax function is typically used to transform output logits into normalized scores that are often interpreted as probabilities. However, these

softmax-derived values do not represent true probabilities in a Bayesian sense; rather, they are uncalibrated confidence scores without a theoretical grounding in probability theory [148]. This shortcoming becomes particularly evident when models are confronted with inputs that differ from their training distribution, commonly referred to as out-of-distribution (OOD) data [96]. In such cases, neural networks often produce highly confident yet incorrect predictions, lacking any mechanism to signal elevated uncertainty or to acknowledge that an input lies outside their domain of competence.

A theoretical framework to equip neural architectures with this capability is offered by Bayesian statistics, giving rise to *Bayesian neural networks (BNNs)* [28], [189]. By treating model parameters as probability distributions rather than fixed values, BNNs yield predictive distributions that quantify the uncertainty associated with each prediction, providing a principled measure of model confidence. Such uncertainty estimates enable more informed decision-making, improve robustness under distributional shift, and are essential for deploying neural networks in safety-critical applications. However, BNNs are considerably more computationally demanding than conventional neural networks, since both the estimation of parameter distributions during training and the computation of predictive uncertainty during inference require multiple stochastic forward passes. This repeated sampling makes their use on embedded and energy-constrained systems particularly challenging.

Consequently, the central question motivating this work is how to realize *resource-efficient and reliable* neural network inference that combines high computational efficiency with trustworthy predictive behavior, encompassing both deterministic and probabilistic models.

This work approaches the goal of resource-efficient inference from two complementary directions. The first focuses on *reducing the computational workload through algorithmic simplification*, achieving comparable performance with fewer operations through techniques such as model compression and probabilistic approximation. The second addresses the *efficient realization of computations* by improving execution efficiency on existing hardware through optimized mapping of computations to hardware resources and by employing specialized architectures—such as analog matrix–multiply accelerators—that execute fundamental operations with much higher energy efficiency. These two directions

are closely connected: algorithmic simplifications can ease hardware demands, while hardware characteristics in turn influence algorithm design and the need for robustness. This work therefore applies this combined perspective first to deterministic neural networks, developing methods for automatic compression on embedded processors and, in parallel, analyzing the execution characteristics of analog matrix–multiply accelerators. It then extends this approach to Bayesian neural networks, where Bayesian approximations, efficient operator–to–hardware mappings, and photonic computing architectures are explored to realize scalable and uncertainty-aware inference. Together, these studies support a unified view of neural network design in which algorithms, compilers, and hardware are co-optimized as elements of a single integrated system. Building on this integrated perspective, the following discussion examines how efficiency and reliability can be advanced in practice.

A well-established approach to enhancing efficiency is the simplification of neural networks through compression techniques. Methods such as quantization [120], [141], pruning [193], [239], [244], and knowledge distillation [194]—discussed in more detail in Chapter 3—are widely used to reduce model size, latency, and power consumption in embedded systems. In practice, it is often unclear how much compression an individual layer can tolerate and how these choices translate into measurable performance gains on the target hardware; consequently, conventional compression pipelines typically apply uniform compression policies across layers, leaving a large degree of layer-specific flexibility unused. Building on this groundwork, *Galen* [21] introduces an automatic, hardware-aware compression framework that combines layer-wise sensitivity analysis with measured latencies on embedded processors. By adapting compression strength per layer and per hardware architecture, Galen leverages this flexibility to generate heterogeneous pruning and quantization strategies optimized jointly for accuracy preservation and inference speed.

Beyond reducing the computational cost of neural network inference, further efficiency gains can be achieved through alternative computing paradigms. Analog accelerators promise substantial improvements in performance per watt, as they perform the fundamental multiply–accumulate (MAC) operation of neural networks at orders of magnitude lower energy than digital implementations [46]. This advantage arises from executing computation directly in the physics of the device rather than through digital abstraction—for example, by charge

accumulation in electrical CMOS-based systems [49] or by optical interference in photonic circuits [163]. However, analog computation is inherently affected by circuit-level imperfections such as nonlinearities, saturation effects, leakage, crosstalk, and various sources of noise. Moreover, these characteristics vary across devices due to manufacturing variations and evolve over time under the influence of environmental factors such as temperature changes. Unlike digital processors, analog hardware does not abstract away these imperfections, and their effects accumulate during computation, requiring algorithms that can tolerate or adapt to them. To achieve such adaptation, neural networks are often trained directly on the hardware in a process known as hardware-in-the-loop training, where forward passes are executed on the physical device to expose the model to real analog imperfections. While this approach effectively compensates for device-specific distortions, its practicality is often limited by the throughput and availability of the hardware. To better understand these effects and to accelerate this process, this work develops a white-box model of analog neural accelerators [44], exemplified by the BrainScaleS-2 (BSS-2) [49] platform, which serves as a representative analog matrix–multiply accelerator. During this modeling process, it was observed that arithmetic operations implemented in analog hardware do not generally preserve associativity [4], a fundamental property assumed in mathematics and relied upon across all domains of computer science—discussed in more detail in Chapter 5.

Building on these models, this work systematically investigates the robustness of neural networks to analog nonidealities through controlled noise injection. The resulting *Walking Noise* framework [9] provides a methodology to map layer-wise sensitivity to additive and multiplicative noise, enabling the identification of particularly robust or vulnerable layers and offering insight into how neural networks learn to tolerate such disturbances through noisy training. In a comparative study of hardening strategies [7], various approaches were evaluated to improve the robustness of neural networks against noisy computations. Among them, noisy training proved to be the most effective technique, but only when the noise characteristics during training were consistent with those encountered during inference—a condition that is difficult to maintain in systems subject to dynamically changing factors such as temperature-induced drift. To address this challenge and further improve resilience under realistic operating conditions, *Variance-Aware Noisy Training (VANT)* [8] introduces adaptive noise scaling

that enables stable performance under environmental fluctuations. Together, these results demonstrate that robustness to analog imperfections requires incorporating hardware characteristics and noise behavior directly into the training process rather than compensating for them after deployment.

While these approaches focus on efficient and reliable execution of deterministic models, many applications also require principled uncertainty estimation to quantify the reliability of predictions. Bayesian neural networks address this by representing network weights as probability distributions, producing predictive posteriors whose variance decomposes into epistemic and aleatoric uncertainty [12], [28]. However, exact Bayesian inference is computationally prohibitive: sampling-based methods such as Markov chain Monte Carlo (MCMC) scale poorly, and even variational approaches like stochastic variational inference (SVI) require repeated stochastic forward passes.

This work therefore investigates Bayesian approximation methods and their hardware-efficient implementation to enable BNN inference on resource-constrained embedded systems. First, the training of BNNs using MCMC and SVI methods is examined, highlighting that the choice of activation function strongly influences convergence, predictive accuracy, and uncertainty estimation.

Second, the *Probabilistic Forward Pass (PFP)* [186] is employed as a closed-form approximation of SVI in BNNs. By assuming both weights and activations to follow Gaussian distributions, it enables analytic propagation of means and variances through all layers of the network, including nonlinearities handled via moment-matching. This removes the need for repeated sampling, replacing multiple stochastic forward passes with a single analytic forward computation. To enable practical deployment, a dedicated operator library implementing these Gaussian-propagating operations is integrated into the Tensor Virtual Machine (TVM) [115] compiler framework and optimized for embedded ARM processors, achieving speedups of several orders of magnitude over naive SVI-based BNN inference [3].

Complementary to this analytically grounded approach, ensemble-based methods offer a more practical route to uncertainty estimation by approximating Bayesian diversity through multiple deterministic predictors. Finally, ensemble-based methods for uncertainty estimation are compared, including *Monte Carlo Dropout (MCDO)* and *Deep Ensembles (DEs)*, alongside the recently introduced *Repulsive Last-Layer Ensembles (RLLEs)* [13]. RLLEs share a common deter-

ministic backbone and replace the final layer with an ensemble of prediction heads, greatly reducing the number of trainable parameters while maintaining prediction diversity. A function-space repulsion term promotes diversity between ensemble heads, enabling efficient fine-tuning on pretrained networks and producing calibrated uncertainty estimates at minimal additional computational cost. Implemented and evaluated using the TVM compiler stack, RLLEs achieve the fastest inference among all evaluated approaches. Together, these studies demonstrate how algorithmic approximations and hardware-aware mapping can jointly reduce the computational cost of BNN inference while maintaining reliable uncertainty estimation.

Similar to the investigation of deterministic neural networks, this work extends the exploration beyond algorithmic approximations and optimizations of established architectures toward emerging analog hardware technologies. In photonic accelerators, intrinsic fluctuations—such as photon shot noise and chaotic light dynamics—introduce stochasticity that is typically regarded as a limitation to precision. Here, this physical noise is leveraged as a controllable source of randomness for probabilistic inference, allowing stochastic sampling to emerge directly from the hardware. In the presented photonic experiments [1], [10], a probabilistic model of the photonic hardware is integrated into Bayesian training and inference loops, enabling the constructive use of stochastic fluctuations characteristic of chaotic light for probabilistic computation. To make this possible, a controllable noise encoding scheme was developed that maps desired activation variances onto optical intensity distributions following Bose–Einstein statistics. The BNN architecture was co-designed with the photonic prototype to account for its physical constraints enabling stable training and reliable uncertainty estimation under real hardware conditions. This approach extends the algorithmic–hardware co-design principle beyond robustness toward the deliberate utilization of analog noise as a foundation for efficient probabilistic computation.

Contributions

This thesis makes the following primary contributions:

- **Compressing DNNs for efficient inference on embedded systems:** This contribution advances resource-efficient deterministic inference through both analytical insight and automated optimization. A comprehensive study of compression methods [12]—including quantization, pruning, and knowledge distillation—demonstrates that reductions in parameters or FLOPs do not reliably predict runtime efficiency, emphasizing the need for empirical, hardware-specific evaluation. Building on this foundation, the *Galen* framework [21] performs automatic, reinforcement-learning-based compression that adapts pruning and quantization per layer and per compute architecture, guided by *sensitivity analysis* and *hardware-in-the-loop* latency measurements, achieving superior accuracy–efficiency trade-offs.

- **Modeling and mitigating analog nonidealities in neural accelerators:** This work addresses the challenges of analog imperfections in neural hardware through complementary advances in modeling and robustness. Detailed white-box and data-driven models of the BSS-2 analog accelerator [4], [44] capture device-specific nonlinearities, saturation effects, noise, and ordering dependencies, revealing the non-associativity of analog accumulation and enabling efficient, hardware-aware training without continuous device access. Building on this foundation, the *Walking Noise* framework [9] introduces a methodology to quantify layer-wise robustness to additive and multiplicative disturbances, while a systematic comparison of hardening strategies [7] reveals noisy training as the most effective approach to improve robustness against noisy analog computations. To further sustain resilience under dynamically varying conditions, such as temperature-induced drift, *VANT* [8] extends this approach through adaptive noise scaling, demonstrating that robust analog computation requires integrating hardware characteristics directly into the training process.

- **Efficient BNN inference on embedded systems:** This work addresses the computational challenges of Bayesian neural networks by developing scalable approximation and implementation techniques that enable efficient inference on embedded hardware. It begins by analyzing BNNs and their core inference methods, MCMC and SVI, evaluating their scalability and uncertainty estimation quality, and revealing how activation functions critically affect convergence and predictive behavior. Building on these insights, the *Probabilistic Forward Pass* [186] is implemented as a closed-form approximation to variational inference, analytically propagating Gaussian means and variances through neural layers. A dedicated operator library integrated into the TVM compiler enables optimized execution on embedded ARM processors, achieving substantial speedups over sampling-based approaches [3]. In addition to these analytic methods, ensemble-based Bayesian approximations are explored, including *Repulsive Last-Layer Ensembles* [13], which promote diversity between prediction heads via a function-space repulsion term while sharing a common deterministic backbone. Complementing the analytic acceleration achieved by the Probabilistic Forward Pass, the TVM-based implementation of RLLEs attains further substantial speedups—particularly at larger batch sizes—while maintaining high-quality uncertainty estimation, establishing RLLEs as an efficient and scalable method for embedded BNN inference.

- **Harnessing photonic noise for probabilistic inference:** This work explores photonic accelerators as a hardware platform for efficient Bayesian neural network inference, leveraging intrinsic optical noise as a controllable source of stochasticity. Through modeling and hardware–algorithm co-design of chaotic-light photonic systems [1], [10], the inherent randomness of light interference is transformed from a limiting factor into a functional entropy source for probabilistic computation. A dedicated noise-encoding scheme and co-adapted BNN architecture enable stable training and accurate uncertainty estimation under real hardware constraints, transforming analog noise from a source of error into an active computational resource.

Publications

The work builds upon and extends the following publications:

Journal Articles

- Wolfgang Roth*, Günther Schindler*, Bernhard Klein*, Robert Peharz, Sebastian Tschiatschek, Holger Fröning, Franz Pernkopf, and Zoubin Ghahramani, "Resource-Efficient Neural Networks for Embedded Systems," *Journal of Machine Learning Research (JMLR)*, 2024. [JMLR] [arXiv]

- Frank Brückerhoff-Plückelmann, Hendrik Borras, Bernhard Klein, Akhil Varri, Marlon Becker, Jelle Dijkstra, Martin Brückerhoff, C. David Wright, Martin Salinga, Harish Bhaskaran, Benjamin Risse, Holger Fröning, and Wolfram Pernice, "Probabilistic Photonic Computing with Chaotic Light," *Nature Communications*, 2024. [DOI]

- Frank Brückerhoff-Plückelmann, Anna P. Ovvyan, Akhil Varri, Hendrik Borras, Bernhard Klein, Lennart Meyer, C. David Wright, Harish Bhaskaran, Ghazi Sarwat Syed, Abu Sebastian, Holger Fröning, and Wolfram Pernice, "Probabilistic Photonic Computing for AI," *Nature Computational Science*, May 2025. [DOI]

Conference Papers

- Hendrik Borras*, Bernhard Klein*, and Holger Fröning, "Walking Noise: On Layer-Specific Robustness of Neural Architectures Against Noisy Computations and Associated Characteristic Learning Dynamics," in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*, 2024. [DOI] [arXiv]

- Xiao Wang, Hendrik Borras, Bernhard Klein, and Holger Fröning, "Variance-Aware Noisy Training: Hardening DNNs Against Unstable Analog Computations," in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*, 2025. [DOI] [arXiv]

Workshop Papers

- Torben Krieger*, Bernhard Klein*, and Holger Fröning, "Towards Hardware-Specific Automatic Compression of Neural Networks," *AAAI Workshop on Practical Deep Learning in the Wild*, 2023. *Best Paper Award.* [DOI]

- Lisa Kuhn*, Bernhard Klein*, and Holger Fröning, "On the Non-Associativity of Analog Computations," in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD) Workshops (ITEM)*, 2025. [DOI] [arXiv]

- Sophie Steger, Christian Knoll, Bernhard Klein, Holger Fröning, and Franz Pernkopf, "Function-Space Diversity for Uncertainty Prediction via Repulsive Last-Layer Ensembles," *ICML Workshop on Structured Probabilistic Inference & Generative Modeling*, 2024. [Link] [arXiv]

- Hendrik Borras*, Bernhard Klein*, and Holger Fröning, "Walking Noise: Understanding Implications of Noisy Computations on Classification Tasks," *HiPEAC Workshop on Accelerated Machine Learning (AccML)*, 2023. (Earlier version of the ECML-PKDD 2024 paper.) [Link]

- Xiao Wang, Hendrik Borras, Bernhard Klein, and Holger Fröning, "On Hardening DNNs Against Noisy Computations," *HiPEAC Workshop on Accelerated Machine Learning (AccML)*, 2025. [arXiv] [Link]

- Bernhard Klein, Christoph Gratl, Manfred Mücke, and Holger Fröning, "Understanding Cache Boundness of ML Operators on ARM Processors," *HiPEAC Workshop on Accelerated Machine Learning (AccML)*, 2021. [arXiv]

- Bernhard Klein, Lisa Kuhn, Johannes Weis, Arne Emmel, Yannik Stradmann, Johannes Schemmel, and Holger Fröning, "Towards Addressing Noise and Static Variations of Analog Computations Using Efficient Retraining," in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD) Workshops (ITEM)*, 2021. [DOI]

Submitted for Review

- Bernhard Klein, Falk Selker, Hendrik Borras, Sophie Steger, Franz Pernkopf, and Holger Fröning, "Accelerated Execution of Bayesian Neural Networks Using a Single Probabilistic Forward Pass and Code Generation," under review at *ACM Transactions on Architecture and Code Optimization (TACO)*, 2025.

* Shared first authorship.

Thesis Structure

The remainder of this thesis is organized into two parts that together advance the goal of resource-efficient and reliable neural network inference.

Part I focuses on deterministic models and explores both algorithmic and hardware-oriented strategies for efficient execution. Chapter 3 surveys established compression techniques and analyzes their effectiveness across embedded computing platforms. Building on these foundations, Chapter 4 presents the *Galen* framework for automatic, hardware-aware compression that jointly optimizes pruning and quantization for accuracy and latency. Chapters 5 and 6 jointly address analog neural accelerators. Chapter 5 develops hardware models that capture device imperfections of analog accelerators, providing a foundation for efficient hardware-aware training. Building on these models, Chapter 6 investigates the resilience of neural networks to such noisy computations, introducing the *Walking Noise* framework for layer-wise robustness analysis and *VANT* to harden DNNs for dynamic noise conditions.

Part II extends these principles to probabilistic machine learning, emphasizing scalable Bayesian inference and uncertainty estimation. Chapter 7 introduces Bayesian neural networks and compares inference methods such as MCMC and SVI, highlighting how activation functions influence uncertainty and convergence. Chapter 8 implements the Probabilistic Forward Pass as a compiler-integrated operator library for embedded inference, while Chapter 9 investigates ensemble-based approximations, including the Repulsive Last-Layer

Digital Hardware · Deterministic	**Digital Hardware · Probabilistic**
Chapter 3: Resource-Efficient Inference Chapter 4: Automatic Model Compression	Chapter 7: Bayesian Neural Networks Chapter 8: Compiling PFP-based BNNs Chapter 9: Ensemble-based BNNs
Analog Hardware · Deterministic	**Analog Hardware · Probabilistic**
Chapter 5: Modeling Analog Hardware Chapter 6: Robustness Against Noisy Computations	Chapter 10: Probabilistic Photonic Computing for BNNs

Figure 1.1 Thesis contributions organized along hardware technologies (vertical: digital vs. analog) and modeling paradigms (horizontal: deterministic vs. probabilistic).

Ensembles, demonstrating their efficiency and calibration trade-offs. Finally, Chapter 10 explores photonic accelerators as a hardware platform for probabilistic computation, showing how intrinsic optical noise can serve as a controllable stochastic resource for BNN inference.

Chapter 11 concludes the thesis by summarizing the overarching findings, synthesizing deterministic and probabilistic efficiency concepts, and outlining future research directions in integrated algorithm–hardware co-design for energy-efficient machine learning.

The overall organization of this work is summarized in Figure 1.1. The diagram arranges the chapters along two axes—hardware domain (digital to analog) and modeling paradigm (deterministic to probabilistic)—mirroring the transition from Part I to Part II. Together, they illustrate how the thesis progresses from digital model compression and analog robustness toward probabilistic and photonic computation as complementary routes to efficient and reliable neural inference.

2
Background

2.1 Deep Neural Networks

Deep neural networks have become the predominant model class in modern machine learning [176], [198]. They consist of multiple layers of parameterized transformations, combined with nonlinear activation functions, that together approximate highly complex mappings from inputs to outputs. The learnable parameters, often referred to as *weights*, are adjusted during training such that the network minimizes a given loss function on available data.

Layer Types. At their core, DNNs alternate between linear operations and nonlinear activation functions. Common choices for the activation function $\sigma(\cdot)$ include the Rectified Linear Unit (ReLU), sigmoid, or hyperbolic tangent, each shaping the representational capacity of the network differently. The simplest linear operation is the dense (fully-connected) layer, which maps an input vector $\mathbf{x} \in \mathbb{R}^d$ to an output vector $\mathbf{y} \in \mathbb{R}^m$:

$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{2.1}$$

where $\mathbf{W} \in \mathbb{R}^{m \times d}$ and $\mathbf{b} \in \mathbb{R}^m$ are learnable parameters. Convolutional layers generalize this operation to exploit the spatial structure of input data while reducing the number of parameters. Let $\mathbf{I}$ denote the input feature map with C channels, and let $\mathbf{W}$ denote a set of filter kernels with spatial dimensions $R \times S$ and U output channels. For an input position (x, y), input channel z and output channel u, the output feature map $\mathbf{O}$ is computed as

$$O[z][u][x][y] = \sum_{k=0}^{C-1} \sum_{i=0}^{R-1} \sum_{j=0}^{S-1} I[z][k][sx+i][sy+j] \cdot W[u][k][i][j] + B[u], \tag{2.2}$$

where $B[u]$ denotes the bias term for output channel u and s is the stride parameter controlling the step size. The resulting output dimensions are

$$E = \frac{H - R + s}{s}, \tag{2.3}$$

$$F = \frac{W - S + s}{s}, \tag{2.4}$$

for an input feature map of height H and width W. This formulation reflects the convolution operation as implemented in modern deep learning frameworks, including multi-channel inputs, multiple filters, stride, and biases.

Training and Inference. Neural networks are trained by gradient-based optimization methods, most prominently stochastic gradient descent and its variants [197], [252]. Gradients are computed efficiently using backpropagation [246], which recursively applies the chain rule across the layered structure. Once trained, inference corresponds to applying the learned transformations to unseen inputs, requiring only the forward pass. This distinction is particularly important in practice: training is usually carried out on large-scale compute clusters, while inference is also often executed on embedded, mobile or specialized hardware where resources are constrained.

Impact and Limitations. The scalability and expressiveness of DNNs have driven breakthroughs across domains such as computer vision, natural language processing, and speech recognition [168], [215]. At the same time, the growing depth and parameter counts of state-of-the-art models result in high computational demand and memory consumption. Balancing prediction quality with computational and energy efficiency is therefore a central challenge, motivating

the resource-aware, robust, and probabilistic inference approaches that form the core contributions of this work.

2.2 ML Frameworks and Compilers

2.2.1 Training Frameworks

Modern deep learning practice is dominated by frameworks such as PyTorch [97], TensorFlow [173], and JAX [112]. They provide user-friendly abstractions to define models, automatic differentiation to compute gradients, and efficient utilization of hardware accelerators. These frameworks have become the standard entry point for designing and training deep neural networks, offering both flexibility for research and scalability for large-scale production.

2.2.2 Probabilistic Programming Frameworks

Classical machine learning frameworks are extended by Probabilistic Programming Languages (PPLs) to model random variables directly. They provide support for sampling, probability distributions, and a range of Bayesian inference algorithms, including variational inference and Markov chain Monte Carlo (see Chapter 7 for a detailed introduction to Bayesian inference methods). Prominent PPLs include Pyro [82], NumPyro [98], and TensorFlow Probability [146].

Pyro is built on top of PyTorch and emphasizes scalable variational inference, making it a natural choice for amortized inference and stochastic variational inference (SVI). NumPyro, by contrast, builds on JAX and leverages just-in-time (JIT) compilation to achieve high performance. It places a stronger emphasis on MCMC methods, providing highly optimized implementations of Hamiltonian Monte Carlo and the No-U-Turn Sampler [206]. In practice, NumPyro is often considered faster for large-scale MCMC-based Bayesian inference due to its compilation-based workflow, while Pyro is regarded as more flexible and easier to customize, making it well suited for prototyping new probabilistic models and inference algorithms. TensorFlow Probability extends TensorFlow with a large collection of probabilistic building blocks, covering both inference methods and probabilistic layers.

These frameworks are a necessary component of the modern probabilistic deep learning ecosystem, providing the modeling tools required to train Bayesian neural networks with uncertainty-awareness while remaining compatible with standard deep learning workflows.

2.2.3 Deep Learning Compilers

Training frameworks excel at model specification and optimization, but efficient deployment on heterogeneous targets often requires compiler support. Hand-tuned vendor libraries deliver strong performance for common operators on popular hardware architectures, yet they are difficult to retarget and cannot cover all combinations of hardware, data layouts, and custom operators. Machine learning compilers address this gap by lowering high-level models to hardware-aware implementations while applying graph- and kernel-level optimizations that improve latency and energy efficiency across diverse backends.

MLIR and XLA. The Multi-Level Intermediate Representation (MLIR) project provides a modular, multi-level intermediate representation with extensible *dialects* for tensors, linear algebra, and device backends [45]. Progressive lowering enables reuse of transformations across ecosystems: models can start from framework-level intermediate representations (IRs), pass through domain-specific dialects, and end as target Instruction Set Architecture (ISA) or runtime code. This design eases support for non-standard data types, dynamic shapes, and emerging accelerators without invasive changes to front-ends. XLA [165] is a domain-specific compiler stack developed for TensorFlow and JAX, providing graph fusion, buffer reuse, and code generation. It remains the primary compiler for Google's TPUs and increasingly relies on MLIR as its underlying compiler infrastructure. Together, MLIR and XLA illustrate how compiler infrastructures can unify optimization passes across frameworks and backends while enabling efficient execution on specialized hardware.

TVM. The Tensor Virtual Machine (TVM) stack is an end-to-end optimizing compiler for deep neural networks that targets CPUs, GPUs, FPGAs, and custom accelerators [115]. Its IR family (TE/TensorIR/Relax) separates *what* to compute from *how* to schedule it. Crucially, TVM is designed to support *custom* operators and *novel* hardware where vendor libraries are unavailable or insufficient, making

it suitable for research settings and embedded deployments alike. In this work, TVM plays a central role across several chapters, serving as the common path from high-level models to hardware-optimized binaries.

Auto-tuning. To illustrate the scheduling challenge, consider the example of a 2D convolution kernel. Its performance depends strongly on how the nested loops over output channels, spatial positions, and kernel indices are ordered, tiled, vectorized, and parallelized. Unfavorable choices—such as iterating over strided dimensions or very small kernels like 3×3 in the innermost loop—can prevent effective vectorization and waste memory bandwidth, while good schedules expose cache locality, SIMD efficiency, and balanced parallel work. Since the optimal decisions vary across tensor shapes and hardware, manual scheduling is tedious and hardware-specific. To address this challenge, TVM integrates auto-tuning frameworks—first Ansor [79] and later the Meta-Scheduler [32]—that automatically explore the scheduling space by generating candidates, benchmarking them on the target device, and learning from results. This process often discovers implementations that rival or surpass expert-crafted kernels, and in this work proved particularly effective for compressed and probabilistic operators where no vendor libraries exist.

Summary. Training frameworks, probabilistic programming environments, and deep learning compilers together form the toolchain that spans the full workflow. Within this ecosystem, TVM offers end-to-end code generation and powerful auto-tuning to achieve high performance for standard and non-standard operators on heterogeneous targets. These capabilities are central whenever hand-tuned kernels are unavailable or suboptimal, enabling efficient deployment for neural network compression (Chapter 4), probabilistic inference (Chapter 8), and ensembles (Chapter 9) on resource-constrained embedded hardware.

2.3 Hardware Platforms for Neural Networks

Advances in hardware have been a key driver of modern deep learning. Both training and inference impose high computational demands, and the choice of hardware platform often determines whether an application is feasible. This section provides a short overview of the most relevant processor classes for

deep neural networks, highlighting their strengths and limitations for inference. All platforms appear in a wide range of scales, from mobile to data-center deployments.

CPUs. Central Processing Units (CPUs) maximize general-purpose programmability and single-thread performance. Modern designs rely on multithreading and vectorization to achieve throughput, which makes them flexible and particularly suitable for handling irregular workloads. Support for low-precision arithmetic (e.g., 8-bit integer instructions) further improves efficiency in typical inference pipelines.

GPUs. Graphics Processing Units (GPUs) contain large numbers of lightweight cores with high memory bandwidth, offering massive throughput for dense tensor operations. They follow a block-parallel execution paradigm, where many threads are grouped into warps and thread blocks, enabling massive fine-grained parallelism [226]. This design makes GPUs highly efficient for regular, highly parallel computations but less effective on irregular workloads. Their native support for reduced-precision arithmetic, such as 16-bit floating point and 8-bit integer, has been central to the deployment of modern deep learning, and recent architectures even extend this support to 4-bit formats for further efficiency gains. Efficient hardware support for dense matrix multiplications has made GPUs the main driver of modern deep neural networks during both training and inference.

FPGAs. Field-Programmable Gate Arrays (FPGAs) consist of configurable logic blocks that can be programmed into custom compute units. They operate at lower frequency and with limited on-chip memory compared to CPUs and GPUs, but allow arbitrary data formats and fine-grained parallelism. Their flexibility makes them attractive for latency-critical deployments with strict real-time constraints, where extreme low inference latencies are required. Moreover, FPGAs are often employed as prototyping platforms for Application-Specific Integrated Circuits (ASICs), offering a balance between programmability and the ability to evaluate custom hardware designs.

Domain-Specific Accelerators. Dedicated accelerators such as Google's TPU [20] implement systolic arrays optimized for dense linear algebra. They

achieve high utilization on structured workloads with limited programmability. Support is typically restricted to standard low-precision formats such as 16-bit floating point or 8-bit integer, but their throughput and energy efficiency make them attractive for large-scale deployments.

Analog Accelerators. Beyond digital processors, analog computing devices have gained attention as promising alternatives for neural network inference. Examples include resistive memory arrays [129], electronic CMOS-based systems [46], [49], and optical or photonic processors [1], [123], [163]. By carrying out matrix multiplications directly in the analog domain, these devices promise orders-of-magnitude improvements in energy efficiency compared to digital hardware. However, analog computations are inherently noisy due to device variability, nonlinearities, and stochastic physical processes [4], [44]. This variability poses challenges for maintaining accuracy, but also creates opportunities for algorithmic co-design. In this work, analog noise and variability are analyzed through analog hardware models (Chapter 5), addressed with robustness-enhancing training methods (Chapter 6), and eventually exploited as a source of stochasticity for probabilistic inference on photonic accelerators (Chapter 10).

Part I

Accelerating Deep Neural Networks

3

Foundations of Resource-Efficient Inference of NN for Embedded Systems

Any intelligent fool can make things bigger and more complex...
It takes a touch of genius—and a lot of courage—
to move in the opposite direction.

— E. F. Schumacher, *Small is Beautiful* (1973)

Modern machine learning has advanced in parallel with the increasing availability of large-scale computational resources. Training and tuning contemporary DNNs is both computationally intensive and methodologically challenging, often requiring massive parallelism and sophisticated systems engineering. State-of-the-art models are typically trained in data centers equipped with abundant GPUs or specialized neural processors (NPUs, TPUs), where energy consumption and latency are secondary considerations. Deployment scenarios, however, present a fundamentally different reality. Embedded systems such as smartphones, autonomous robots, or sensor nodes in the Internet of Things operate under strict resource budgets: available memory is limited, energy is constrained by batteries,

and latency must remain within tight bounds to ensure real-time responsiveness.

This discrepancy creates a persistent tension: how can models that are developed and trained in nearly unconstrained laboratory environments be deployed on hardware with orders of magnitude fewer resources? Naïve model reduction often leads to unacceptable drops in prediction quality. Instead, resource-efficient inference requires systematic approaches that balance representational and computational demands against accuracy in a controlled fashion.

Among the wide variety of proposed strategies, three broad categories have proven particularly effective for DNNs compression and acceleration: *quantization*, *pruning*, and *structural efficiency*. These techniques reduce the memory footprint and inference cost while preserving competitive predictive performance. They are relevant across embedded hardware platforms ranging from general-purpose CPUs and GPUs to domain-specific accelerators and FPGAs. At the same time, they differ markedly in terms of hardware friendliness, achievable compression ratios, and their impact on prediction quality. As we will see throughout this chapter, the effectiveness of compression techniques is highly hardware-dependent, underscoring the need for co-design of algorithms and deployment platforms.

This chapter builds on our work Roth, Schindler, Klein, Peharz, Tschiatschek, Fröning, Pernkopf, and Ghahramani – *Resource-Efficient Neural Networks for Embedded Systems* [12], which provides a comprehensive survey of embedded hardware platforms and efficiency-enhancing methods. For details beyond the scope of this chapter we refer the reader to the original work, while here we focus on the concepts and techniques most relevant for the subsequent chapters.

By consolidating these foundations, this chapter prepares the ground for the Galen framework introduced in the following chapter, which employs hardware-in-the-loop optimization to automate compression strategies.

3.1 Foundations of Efficiency

Resource-efficient inference of DNNs is naturally a multi-objective optimization problem. On the one hand, the model must fit into the tight resource budgets of embedded systems. On the other hand, it must deliver sufficient prediction quality for the target application. As proposed by Roth, Schindler, Klein, Peharz, Tschiatschek, Fröning, Pernkopf, and Ghahramani [12] and illustrated in

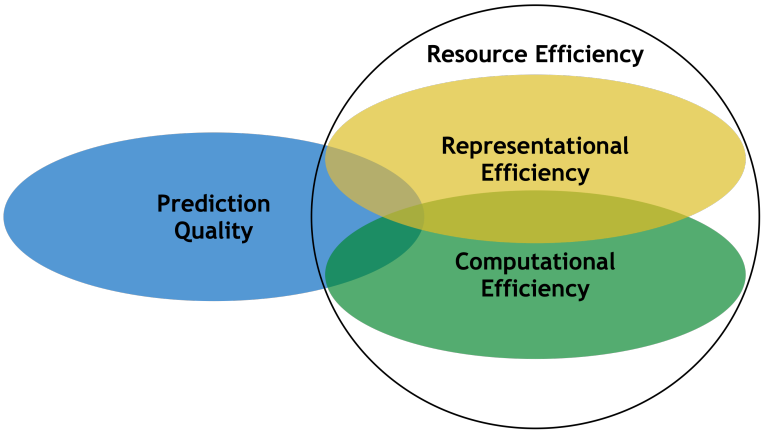


Figure 3.1 Systematic perspective on resource-efficient machine learning. Representational efficiency, computational efficiency, and prediction quality form the triad that structures the discussion in this chapter. Reproduced with permission from [12].

Figure 3.1, efficiency can be structured into three complementary dimensions: representational efficiency, computational efficiency, and prediction quality.

Representational efficiency captures how compactly parameters and activations are stored, measured by memory footprint and storage format. It is determined by the number of parameters, their sparsity, and the chosen numerical precision, for example floating-point versus quantized integer representations. Techniques such as quantization and pruning directly target this dimension.

Computational efficiency refers to the time, throughput, and energy required to execute the model on the target hardware. It accounts not only for arithmetic operations but also, critically, for the movement of data through the memory hierarchy. Simple theoretical metrics such as FLOPs provide a rough indication of computational cost, but they do not reliably capture actual latency or energy consumption. This underlines the importance of evaluating efficiency directly on the deployed device.

Prediction quality captures the predictive performance of the compressed model. While classical machine learning focuses almost exclusively on accuracy, in embedded deployment prediction quality must be considered jointly with efficiency demands to ensure that gains in efficiency do not come at the expense of essential predictive performance.

Taken together, these three dimensions provide a systematic lens for analyzing and comparing efficiency techniques. Throughout the remainder of this chapter we will use them as a recurring reference point to evaluate quantization, pruning,

and structural efficiency.

3.2 Quantization

Quantization reduces the bit-width used to represent weights and activations in DNNs, shrinking model size and activation memory while, on suitable hardware, also accelerating inference and lowering energy cost. At the extreme, binary weights $w \in \{-1, 1\}$ and activations $x \in \{-1, 1\}$ turn multiplications into efficient XNOR and bitcount operations, effectively reducing a network to a logical circuit.

Training such discrete-valued networks is difficult because quantization is non-differentiable. The central challenge is to lower precision as much as possible without sacrificing the accuracy of a full-precision baseline. Over the past three decades, a wide range of techniques have been developed to address this challenge. The following sections review these approaches, beginning with early attempts at reduced-precision training and moving toward modern quantization-aware training and large-scale applications. Early works such as Höhfeld and Fahlman [240], [241] introduced stochastic rounding to prevent training stalls at low precision, a principle later shown effective on modern architectures [192].

Lin et al. [200] eliminated most multiplications during training by quantizing weights stochastically to binary or ternary values in the forward pass and quantizing activations to powers of two in the backward pass, reducing multiplications to bit shifts. This yields significant training speedups, although inference still depends on full-precision weights.

Subsequent work broadened the scope. Courbariaux et al. [190] systematically compared floating-, fixed-, and dynamic fixed-point formats across bit-widths. Lin et al. [181] cast layer-wise bit allocation as a convex optimization problem that minimizes storage under a signal-to-quantization-noise constraint, yielding closed-form solutions for optimal bit-widths.

3.2.1 Quantization-aware Training

Quantization functions are piecewise constant with zero or undefined derivatives, which breaks standard backpropagation. The straight-through estimator (STE) [211], illustrated in Figure 3.2, has therefore become the default workaround: weights are stored in full precision, quantized in the forward pass,

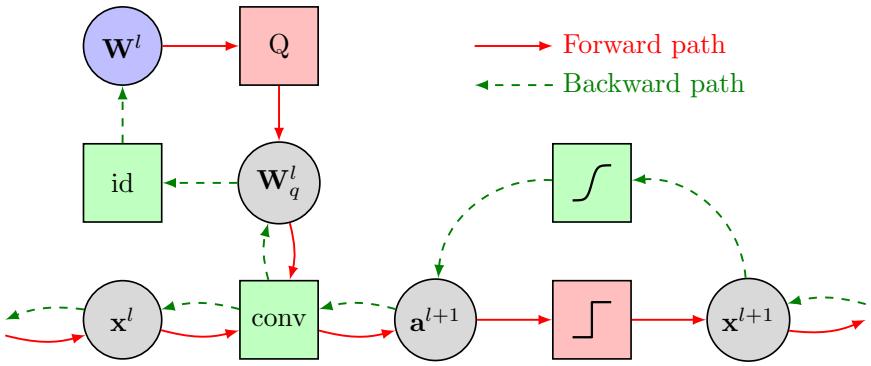


Figure 3.2 Schematic illustration of quantization with the standard deviation. The forward pass applies quantization to weights and activations, while the backward pass approximates the gradient as the identity function. Reproduced with permission from [12].

and updated in the backward pass as if the quantizer were the identity. At test time, only the quantized weights are kept, and the same principle applies to activations.

Early quantization-aware training (QAT) methods applied this scheme to binary networks. Courbariaux et al. [190] trained binary-weight models using either deterministic sign rounding or stochastic rounding via a hard–sigmoid, and Hubara et al. [180] extended the approach to activations, reducing multiplications and additions to efficient XNOR and bitcount operations. Li et al. [30] advanced to ternary weights $w \in \{-a, 0, a\}$ with thresholds chosen to minimize quantization error, while Zhu et al. [171] generalized to asymmetric ternary $w \in \{-a, 0, b\}$ with trainable parameters and per-layer thresholds, often outperforming symmetric schemes.

Filter quantization soon followed. Rastegari et al. [185] approximated convolutional filters as $\mathbf{W} = \alpha\mathbf{B}$, replacing most multiplications with additions and requiring only one multiplication per channel, with input quantization further enabling XNOR and bitcount convolutions. Lin et al. [157] improved accuracy by expressing filters as linear combinations of multiple binary bases.

Other approaches tailored quantizers to data distributions. Cai et al. [145] introduced half-wave Gaussian quantization to better match ReLU activations. Miyashita et al. [184] quantized to powers of two, eliminating multiplications and improving robustness, while Zhou et al. [170] proposed incremental quantization, alternately quantizing subsets of weights to $\{0\} \cup \{2^k\}$ and retraining until all layers were quantized.

Deployment-oriented schemes also emerged. Jacob et al. [121] introduced integer-only inference, emulating quantization during training and deploying 8-bit weights for inference, thereby reducing the model size by a factor of four. Liu et al. [126] presented Bi-Real Net, a ResNet variant with binary convolutions in the residual path and real-valued shortcuts to preserve expressiveness.

More flexible quantizers were later learned end-to-end. Zhang et al. [141] proposed *LQ-Net*, where quantization codebooks are optimized during training, with layer-wise activation quantizers and channel-wise weight quantizers improving flexibility and efficiency. Louizos et al. [95] introduced *Relaxed Quantization*, a differentiable soft-rounding scheme that injects noise before rounding and employs Gumbel–softmax [152] to approximate discrete levels, with a hard STE variant for exact quantization.

These methods illustrate the spectrum from binary to low-bit quantization and highlight the trade-off between efficiency and accuracy. As shown in Figure 3.3, accuracy decreases nonlinearly with lower bit-widths: weight-only quantization impacts performance, but activation quantization leads to larger errors, and combining both amplifies the effect.

Mixed-precision quantization further refined these ideas. Dong et al. [86] ranked network blocks by Hessian eigenvalues to assign bit-widths and defined an order for sequential quantization and fine-tuning with QAT.

In most practical implementations, quantization follows a *linear* scheme, in which the continuous value range is uniformly partitioned into discrete levels. Such linear quantizers are characterized by the quantization step size Q_d, the dynamic range $Q_{\max}$, and the bit-width Q_b, which are related by

$$Q_{\max} = (2^{Q_b - 1} - 1)\, Q_d. \tag{3.1}$$

Esser et al. [60] proposed to make the step sizes Q_d^l trainable and learn them with the STE, in contrast to fixed statistics as in XNOR-Net [185]. Uhlich et al. [74] extended this to mixed precision and compared alternative parameterizations of Eq. (3.1), finding that optimizing $(Q_d^l, Q_{\max}^l)$ by backpropagation yields stable training and implicitly determines the effective bit-widths Q_b^l.

Quantization during backpropagation. Several methods also quantize gradients to further reduce training cost. Zhou et al. [142] proposed flexible

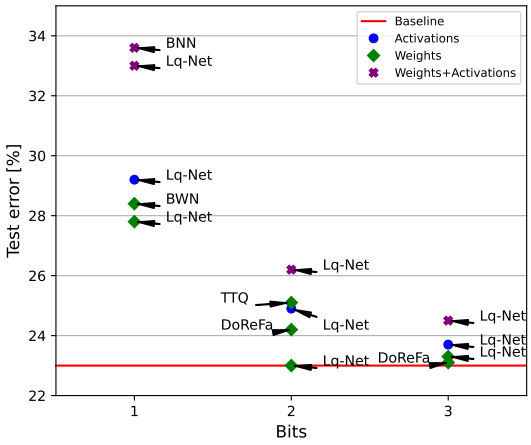


Figure 3.3 Comparison of several popular quantization methods using the DenseNet-BC-100 architecture on the CIFAR-100 dataset. Test error is shown as a function of bit-width for weight and activation quantization. As expected, lower bit-widths lead to larger errors: weight-only quantization degrades accuracy moderately, activation quantization incurs larger losses, and quantizing both amplifies the effect. Reproduced with permission from [12].

bit-width schemes for weights, activations, and backpropagation, highlighting the importance of stochastic quantization. Wu et al. [139] extended this idea with customized quantizers for weights, activations, and their gradients, enabling integer arithmetic throughout training and inference and accumulating updates directly in low precision.

Theory and large-model results. Beyond empirical studies, theoretical work has analyzed convergence and approximation in quantized training [111], [155]. Shekhovtsov and Yanush [70] further showed that the STE can be derived from linearization approximations in stochastic binary networks. Quantization is also increasingly applied to large language models [25], [54], [71], where QAT is often impractical. Frantar et al. [18] addressed this with an efficient one-shot post-training quantization method for transformers, while Lin et al. [11] proposed activation-aware weight quantization that preserves salient weights via activation magnitudes and per-channel scaling.

Most research on quantization emphasizes algorithmic methods and accuracy trade-offs, but hardware studies reveal additional critical factors. In our previous work [43], we showed that matrix multiplication and convolution on embedded ARM CPUs are typically *cache-bound*: performance is limited by L1 bandwidth rather than arithmetic throughput. Quantization can mitigate this by reducing

memory traffic, but the gains depend heavily on data layout and packing overhead, with bit-serial approaches particularly sensitive to bit-width. This analysis highlights that quantization must be co-designed with memory hierarchies and software stacks to achieve real speedups on embedded platforms.

3.2.2 Bayesian Approaches for Quantization

Several quantization methods connect naturally to Bayesian and variational inference. Achterhold et al. [110] extended the variational-dropout pruning approach of Louizos et al. [160] with mixtures of log-uniform priors centered at quantization levels, concentrating the posterior on discrete values and enabling quantization without fine-tuning.

Other works model discrete weight distributions directly. Soudry et al. [210] approximated the posterior with expectation propagation, while Shayer et al. [134] optimized distributions over binary or ternary weights via the local reparameterization trick [196]. Peters and Welling [130] adapted this to sign activations, and Roth and Pernkopf [100] extended it to more than three discrete values.

Van Baalen et al. [52] introduced Bayesian mixed-precision quantization for power-of-two bit-widths, using recursive quantization with gates trained by variational inference; a zero-bit gate simultaneously enabled pruning.

Havasi et al. [89] proposed a Bayesian compression scheme that learns a mean-field variational posterior and encodes samples via importance-sampled atomic approximations, allowing efficient coding and recovery with a shared random seed.

Summary. Quantization is one of the most effective and versatile compression techniques for DNNs. It spans a spectrum from simple 8-bit integer quantization to advanced mixed-precision and Bayesian formulations. Recent progress in learned quantizers, Hessian-aware policies, and Bayesian perspectives has improved robustness and flexibility. At scale, quantization is essential for deploying large models efficiently on both data-center accelerators and resource-constrained embedded systems. Overall, it provides a principled trade-off between efficiency and accuracy and remains a cornerstone of hardware-aware inference across model types and computing platforms.

3.3 Pruning

Pruning reduces the size and cost of DNNs by enforcing parameter sparsity: weights judged unimportant are set to zero, and the resulting sparsity is leveraged for efficiency. Two main paradigms are distinguished. *Unstructured pruning* removes individual weights irrespective of their tensor location, typically preserving accuracy but requiring extreme sparsity and specialized structures to yield speedups. *Structured pruning*, in contrast, eliminates entire neurons, channels, or filters, directly reducing tensor dimensions and remaining compatible with standard dense operations.

The following sections review both unstructured and structured approaches, discuss Bayesian formulations, and conclude with methods that adapt pruning dynamically at inference time.

3.3.1 Unstructured Pruning

Early work on pruning already explored sophisticated techniques beyond simple magnitude thresholds. LeCun et al. [244] introduced *optimal brain damage*, which uses a second-order Taylor expansion with a diagonal Hessian to estimate the loss increase from pruning individual weights. Weights with minimal estimated impact are removed, alternating with retraining to recover performance. Hassibi and Stork [239] refined this idea with *optimal brain surgeon*, which approximates the full covariance matrix to prune low-impact weights while simultaneously adapting the remaining parameters. Although effective on small networks, these methods do not scale to modern architectures with millions of parameters, as computing and inverting the Hessian introduces prohibitive computational and memory costs.

As a result, most later approaches returned to simpler magnitude-based pruning. Han et al. [193] demonstrated that iteratively removing small weights and retraining can shrink networks substantially with little loss in accuracy. Their follow-up, *Deep Compression* [178], extended this idea by combining pruning with quantization, weight sharing, and Huffman coding, achieving reductions of up to 49× in memory footprint and 3–5× in energy use.

Guo et al. [177] argued that pruning should be reversible. They introduced binary masks to track pruned weights and updated them with the STE, allowing

previously removed connections to reappear when beneficial.

More recent work has also explored random sparsity. Gadhikar et al. [19] showed that fixed random pruning patterns can induce effective sparsity at very low cost, making them attractive as initialization masks even if they are not optimal in isolation.

3.3.2 Structured Pruning

Structured pruning removes entire groups of weights—such as neurons, channels, or filters—so that the resulting models remain compatible with dense tensor operations and yield hardware-level efficiency. Mariet and Sra [183] used determinantal point processes to identify diverse, non-redundant neurons, pruning the rest and adjusting outgoing weights to minimize changes in the next layer. Wen et al. [187] incorporated group lasso regularization into training to induce sparsity at the level of filters, channels, or even whole layers. Liu et al. [159] exploited batch normalization, pruning channels by thresholding learned scaling factors, while Huang and Wang [119] introduced trainable scaling coefficients with ℓ^1 regularization, driving entire structures to zero.

Other methods adopt more data-driven criteria. ThiNet [161] pruned channels that minimally affected the output of the subsequent layer, followed by least-squares reconstruction of the remaining activations.

Bayesian perspectives have also influenced structured pruning. Louizos et al. [127] attached stochastic binary gates with trainable probabilities to weights or structures, encouraging sparsity with an ℓ^0 regularizer and enabling differentiable training via Gumbel–softmax relaxations [152]. Li et al. [91] refined this approach using an unbiased gradient estimator [107].

Finally, Liu et al. [94] argued that retraining pruned dense models is not always necessary: training sparse architectures from scratch can achieve comparable or better accuracy, suggesting that pruning is closely related to neural architecture search.

Bayesian Pruning. Bayesian approaches cast pruning as posterior inference over sparse weights. Graves [219] and Blundell et al. [189] applied mean-field variational inference with Gaussian posteriors and pruned weights with low signal-to-noise ratios. Variational dropout [196] provided another route:

Molchanov et al. [162] optimized dropout rates per weight, removing those with high rates. Louizos et al. [160] extended this idea to structured groups using sparsity-promoting priors, linking pruning decisions to bit-width allocation via the minimum description length principle [227].

Summary. Pruning reduces network size and computational cost by eliminating weights or structures with little impact on predictive accuracy. Unstructured methods can achieve very high sparsity but often fail to deliver speedups in practice, while structured pruning removes whole units such as neurons or channels, aligning with dense operations and yielding tangible efficiency gains. Together, these results highlight the heavy over-parameterization of modern networks and establish sparsity as a key mechanism for resource-efficient inference.

3.4 Neural Architecture Search

Neural architecture search (NAS) automates the search for effective DNN architectures. Rather than relying on manual design, NAS explores a search space of candidate architectures and seeks models that optimize validation accuracy while increasingly incorporating resource efficiency objectives. This makes NAS particularly relevant for embedded deployment, where efficiency is as important as accuracy.

The main difficulty lies in the high cost of evaluating candidate architectures, since each requires training and only yields noisy performance estimates, combined with the exponential growth of the search space. Zoph et al. [172] cast NAS as a reinforcement learning task, where a controller RNN generated architectures and received validation accuracy as reward. While effective, this approach required thousands of full training runs, making it impractical at larger scale. Follow-up work reduced cost through proxy tasks, parameter sharing, or differentiable relaxations [143].

Several approaches incorporated hardware-awareness directly into the objective. MnasNet [102] extended reinforcement learning NAS with latency measurements on mobile devices, producing models that respect device-specific runtime constraints. ProxylessNAS [83] avoided proxy tasks by training over-parameterized networks in which each layer contained multiple candidate blocks; block-selection probabilities were optimized by gradient descent with the STE,

and predicted device latencies were included as a differentiable regularizer. Single-pass NAS [101] further simplified this by consolidating all operations into a shared superblock, enabling efficient gradient-based training of both architecture and weights.

EfficientNet [73] represents a widely adopted outcome of NAS. The method first searches for a small, resource-efficient baseline model and then enlarges it systematically using compound scaling of depth, width, and resolution, achieving state-of-the-art performance on ImageNet with comparatively modest model sizes.

Beyond architecture design, NAS principles have also been applied to compression. Wang et al. [104] used reinforcement learning to assign layer-wise bit widths for mixed-precision quantization, incorporating hardware-specific latency and energy constraints estimated from device-specific lookup tables. Similarly, Wu et al. [138] optimized layer-wise quantization gates via differentiable stochastic optimization. These approaches demonstrate how search-based methods can jointly reason about architecture and compression to produce hardware-aware models.

As discussed previously, pruning can also be interpreted as a form of implicit NAS. Liu et al. [94] showed that training sparse networks directly from scratch often matches or surpasses iterative prune–retrain pipelines, suggesting that pruning primarily identifies effective architectures rather than merely compressing weights.

Building on the ideas of quantization, pruning, and NAS, the proposed *Galen* framework [21] extends automatic compression by jointly searching over quantization levels and pruning ratios. The search is driven by reinforcement learning, while hardware-in-the-loop latency measurements and sensitivity analyses provide guidance on efficiency and accuracy. In this way, *Galen* unifies multiple compression strategies within a single, hardware-aware optimization framework. A detailed discussion of this contribution follows in the next chapter.

For a broader overview of NAS, including search spaces, optimization strategies, and performance estimation techniques, see Elsken et al. [87]. Together, these works highlight NAS as a flexible paradigm that can target not only accuracy but also resource efficiency and compression.

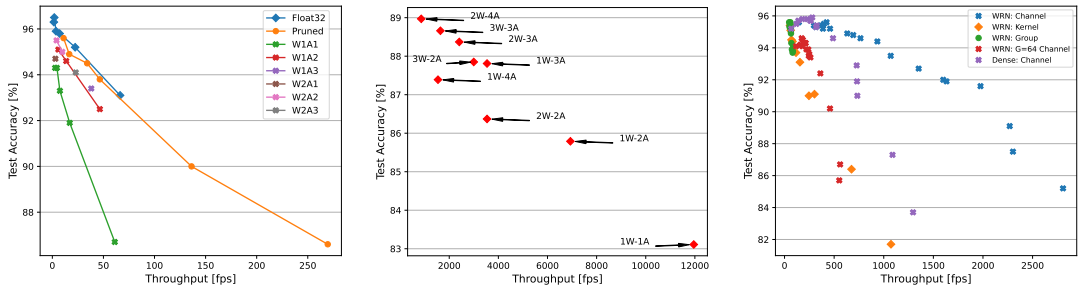
3.5 Hardware Platforms under Compression

The efficiency of compression methods such as quantization and pruning strongly depends on the hardware platform on which the model is deployed. While the general characteristics of CPUs, GPUs, FPGAs, and domain-specific accelerators were introduced in Chapter 2.3, we here highlight how compression interacts with these platforms in practice. The discussion is guided by the results of [12], which compare throughput–accuracy trade-offs across multiple hardware targets.

CPUs. Central Processing Units are highly flexible platforms that can, in principle, support low-bit-width quantization in software as well as unstructured sparsity. In practice, however, performance gains are most pronounced when using datatypes directly supported by the hardware, such as 8-bit integers, together with structured pruning that aligns with SIMD vector units and cache hierarchies. This makes CPUs particularly effective at exploiting structured sparsity, while unstructured pruning or very low-bit quantization typically require extreme compression ratios to compensate for the overhead of irregular memory access and instruction scheduling.

GPUs. Graphics Processing Units benefit from both quantization and structured pruning. Tensor cores and SIMD units efficiently accelerate dense matrix operations and support low-precision arithmetic (e.g., FP16, INT8, and in recent architectures even INT4). Structured pruning can also improve throughput, provided sparsity patterns align with the block-parallel execution paradigm, while unstructured pruning is largely ineffective due to irregular memory access. In practice, both structured pruning and quantization are employed on GPUs, with the relative advantage depending on the specific architecture, workload characteristics, and precision requirements.

FPGAs. Field-Programmable Gate Arrays excel at extreme quantization, as arbitrary data formats and customized compute pipelines can be implemented directly. This makes them particularly suitable for binary or ternary neural networks. Pruning is also supported, but benefits are limited by on-chip memory and routing overhead. In practice, FPGAs are most effective when paired with aggressively quantized models that meet strict real-time constraints.



(a) ARM CPU (b) FPGA (c) Embedded GPU

Figure 3.4 Throughput–accuracy trade-offs of compressed models on CIFAR-10 across embedded hardware platforms: (a) quantized and pruned WRN [188] models on an ARM CPU, (b) quantized VGG [202] models on an FPGA data-flow architecture using FINN [167], and (c) different pruning methods on an embedded GPU. Reproduced with permission from [12].

Domain-Specific Accelerators. Dedicated accelerators such as Google's TPU are optimized for dense linear algebra, often implemented as systolic arrays. They achieve high utilization on structured workloads and natively support accelerator-specific low-precision datatypes such as 16-bit floating point or 8-bit integer, which enable speedups for accordingly quantized models by design. However, these platforms provide little flexibility for arbitrary precisions or fine-grained sparsity, so pruning is generally less effective unless explicitly supported by hardware extensions.

3.6 Evaluation on Embedded Hardware

To complement the survey of compression techniques, we briefly review empirical results of executing compressed models on embedded processors. We compare quantization and pruning across representative embedded devices: an ARM CPU, a reconfigurable FPGA, and an embedded GPU. All devices operate in a comparable power envelope of about 5 W, which enables a fair comparison of efficiency–accuracy trade-offs.

Embedded ARM CPU Figure 3.4 (a) shows throughput–accuracy trade-offs of pruned and quantized WRN [188] models on an ARM Cortex-A53. Both quantization and pruning improve throughput over the baseline, but strong compression in either form reduces accuracy. Structured channel pruning achieves

the highest throughput while maintaining competitive accuracy, reflecting that CPUs exploit sparsity more effectively than low-bit arithmetic.

FPGA On reconfigurable hardware, quantization is essential to fit models into limited on-chip resources. Figure 3.4 (b) shows quantized VGG [202] models on a XILINX Ultra96 FPGA using the FINN framework [167]. Throughput rises sharply at very low bit-widths, but accuracy falls as precision decreases. The Pareto front is reached with extremely low-precision weights (1–2 bits) and moderately higher-precision activations (3–4 bits), confirming that activation quantization dominates predictive accuracy on FPGAs.

Embedded GPU Figure 3.4 (c) reports results for pruned WRN [188] and DenseNet [151] models on a Jetson Nano GPU. All pruning methods improve throughput, but excessive compression reduces accuracy. Channel pruning strikes the best balance of accuracy and efficiency, while group and kernel pruning cut FLOPs without speeding up inference, underscoring that FLOPs are a poor runtime proxy on GPUs. Here, efficiency is more closely tied to reducing activation memory and aligning with the accelerator's software stack.

Cross-Platform Comparison Figure 3.5 summarizes throughput–accuracy trade-offs across CPU, FPGA, and GPU platforms. CPUs offer high flexibility and support fine-grained sparsity, but in this comparison their general-purpose design yields lower throughput than the more parallel accelerator architectures when used together with structured compression. GPUs dominate the high-accuracy, high-throughput regime, benefiting from massive parallelism and large memory. FPGAs achieve outstanding throughput at low precision, but limited on-chip resources often necessitate aggressive quantization that reduces accuracy.

These results highlight that the effectiveness of compression is highly hardware-dependent. While quantization is mandatory on FPGAs, pruning aligns better with CPUs, and GPUs require compression strategies that optimize memory and software support rather than raw FLOPs.

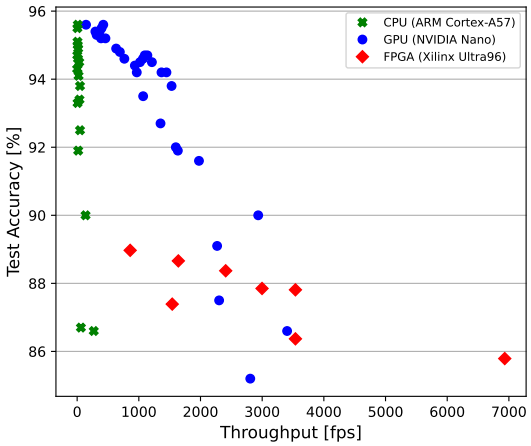


Figure 3.5 Throughput–accuracy trade-offs of compressed models on the CIFAR-10 task across different processor architectures (CPU, FPGA, GPU). GPUs dominate the high-accuracy, high-throughput regime, being well suited for dense and batched matrix multiplications. FPGAs can deliver very high throughputs, but their limited on-chip resources often necessitate aggressive quantization, which reduces predictive accuracy. CPUs offer high flexibility and support fine-grained sparsity, but in this comparison their general-purpose design yields lower throughput than the more parallel accelerator architectures when used together with structured compression. Reproduced with permission from [12].

Summary

This chapter introduced the foundations of resource-efficient inference of deep neural networks on embedded systems. We reviewed three main approaches: quantization, pruning, and neural architecture search, and discussed how their benefits depend critically on the target hardware.

From the survey and experiments, several lessons emerge:

- **Quantization** reduces memory traffic and enables low-precision execution. In image classification, accuracy is particularly sensitive to activation bit-width, while weight quantization is often more benign. The actual benefits, however, depend strongly on memory hierarchies and data layouts.
- **Pruning** exploits over-parameterization to remove redundant weights or structures. Structured pruning yields practical speedups on dense hardware, whereas MAC reductions alone often fail to translate into latency gains.
- **Neural architecture search** provides a framework to automate design

under accuracy–efficiency trade-offs, and recent work has extended these ideas to compression-aware settings.

Hardware evaluations reinforced that compression is not universally effective: CPUs can leverage fine-grained sparsity, FPGAs excel with low-precision quantization, and GPUs demand memory-efficient dense mappings. Crucially, FLOP counts and parameter numbers are poor predictors of real performance; hardware co-design and direct measurements are essential to identify viable efficiency gains.

Together, these observations highlight both the promise and the complexity of resource-efficient inference. While individual techniques succeed under specific conditions, practical deployment requires balancing multiple strategies and aligning them with device characteristics. The next chapter introduces Galen [21], a framework that addresses this challenge by automating the search over pruning and quantization parameters with reinforcement learning and hardware-in-the-loop feedback. Galen builds directly on the lessons of this chapter: rather than applying quantization and pruning in isolation, it integrates them into an automated framework that searches for compression strategies tailored to the constraints of a given hardware platform.

4

Galen: Automatic Model Compression

Perfection is achieved not when there is nothing more to add,
but when there is nothing left to take away.

— Antoine de Saint-Exupéry, *Wind, Sand and Stars* (1939)

In practice, compression is often applied with *global* settings—uniform bit widths and a single sparsity target across all layers—because selecting safe *per-layer* quantization and pruning levels requires expertise in DNN design and compression, as well as in the efficient implementation on the specific hardware architecture. Per-layer compression, by contrast, allocates precision and sparsity at the layer granularity to exploit heterogeneity in hardware payoff. Yet without automation and reliable hardware feedback, this flexibility remains largely unused in deployed systems.

Building on the previous chapter's background on pruning and quantization, we now address the question of *how to automate per-layer compression decisions for a concrete deployment device.* Proxy objectives such as FLOPs, MACs, or BOPs provide only weak guidance for end-to-end latency on real systems, where compiler optimizations, memory hierarchies, parallel execution, and vectorization

dominate runtime.

Galen targets this gap with a hardware-in-the-loop formulation that learns device-specific, layer-wise compression policies [21]. The key idea is to align the objective with actual execution by coupling policy learning to two grounded signals: (i) *measured* on-device latency obtained by compiling candidates with TVM [115] and benchmarking on the target platform, and (ii) *per-layer sensitivity* that quantifies how tolerant each layer is to pruning and quantization. Optimizing against these signals enables the search to prefer policies that are robust in accuracy while improving real latency on the final hardware.

Conceptually, Galen extends the promise of combining compression methods—shown effective in *Deep Compression* [178]—to an automatic, hardware-aware setting. In contrast to prior automatic approaches that optimize a single dimension under proxy cost models (e.g., pruning in AMC [118] or quantization in HAQ [104]), Galen *jointly* learns per-layer policies guided by measured latency and sensitivity cues. This design systematically exploits per-layer heterogeneity and closes the loop between compression decisions and measured on-device latency.

4.1 Automatic Model Compression

A large body of research has sought to automate model compression by framing it as an optimization problem. NetAdapt [140] introduced a platform-aware greedy procedure that iteratively prunes channels and validates the effect on device latency, thereby integrating hardware measurements into the compression loop. Other methods replaced this heuristic search by more sophisticated optimization strategies, such as simulated annealing in AutoCompress [67] or evolutionary algorithms in Automatic Structure Search [66]. While differing in their algorithms, these approaches are similar in that they largely optimize proxy metrics such as FLOPs, MACs, or parameter counts, which only approximate the behavior of the underlying hardware.

In parallel, mixed-precision quantization has been explored as a way to further reduce the cost of inference. Approaches in this line of work differ primarily in how they determine layer-wise bit widths. Some rely on sensitivity analysis to estimate the robustness of layers, with Hessian-aware methods such as HAWQ and HAWQ-V2 being prominent examples [57], [86]. Others instead formulate policy

generation as a learning problem, training reinforcement learning agents that predict quantization strategies, as in AutoQ and ReLeQ [59], [69]. Despite the difference in methodology, these frameworks share a reliance on proxy estimates of latency—derived from look-up tables or analytic surrogates—rather than direct measurements of the target device.

The effectiveness of combining pruning and quantization was first demonstrated by *Deep Compression* [178], which showed that high compression ratios can be achieved without compromising accuracy when multiple techniques are applied in sequence. Building on this insight, later work explored joint optimization formulations, for instance in CLIP-Q [136], differentiable joint pruning–quantization objectives [76], and joint architecture–compression searches such as APQ [75]. These contributions underscore the value of multi-dimensional compression, yet they typically continue to optimize with respect to abstract metrics such as FLOPs or bit operations (BOPs) [35] rather than actual device behavior.

A related but distinct line of research lies in hardware-aware NAS, which aims to design new architectures under hardware constraints. Here, approaches diverge in the way device cost is incorporated. Some works continue to use simple analytic metrics such as FLOPs or parameter counts as a stand-in for latency. Others, such as FBNet, train cost models to predict latency from sampled measurements, thereby providing a differentiable surrogate objective [106]. More recent frameworks, including MnasNet [102] and Once-for-All [55], take a step further by relying on direct measurements on the target device. This progression from abstract metrics to cost models and finally to measured performance mirrors the developments observed in compression research, highlighting the importance of grounding optimization in real device characteristics.

Among these many directions, AMC [118] and HAQ [104] are most closely related to our work. Both adopt a reinforcement learning formulation to derive layer-wise compression policies, where a policy network sequentially decides pruning or quantization actions. While these frameworks demonstrated the feasibility of automated compression, they remain limited to a single dimension—pruning for AMC and quantization for HAQ—and rely on proxy metrics to enforce efficiency, namely FLOPs or parameter counts in AMC, and look-up-table latency estimates in HAQ. In contrast, Galen [21] jointly reasons about pruning and quantization, incorporates layer sensitivity into the optimization, and bases its

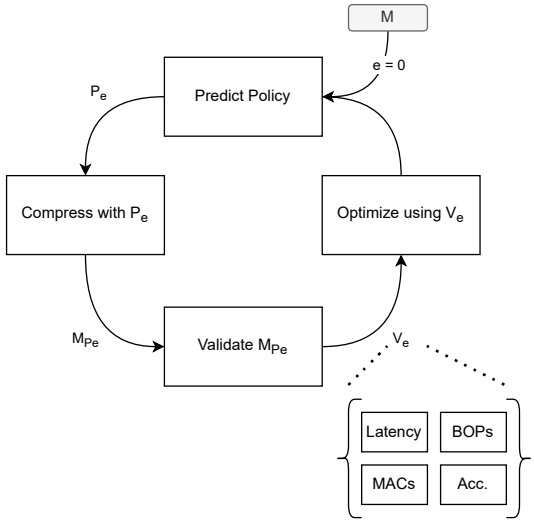


Figure 4.1 Episode overview in *Galen*: (1) predict a compression policy; (2) apply it to obtain the compressed model; (3) evaluate on-device (compile, benchmark, and validate accuracy); (4) update the agent via the reward. Reproduced with permission from [21].

decisions on on-the-fly hardware measurements, thereby closing the gap between proxy objectives and actual device performance.

4.2 Galen Methodology

The goal of *Galen* is to automatically derive hardware-aware compression policies that balance model accuracy and inference latency. To this end, Galen frames compression as a reinforcement learning problem, where an agent incrementally decides how much to prune or quantize each layer of a network. After applying a complete policy, the resulting model is compiled and benchmarked on the target device, and the measured accuracy–latency trade-off serves as feedback for learning. This section describes the algorithmic concept, the formulation of pruning and quantization policies, the design of Galen's agents, and the integration of sensitivity analysis and hardware latency measurements.

4.2.1 Algorithmic Concept

At the center of Galen is the notion of a *compression policy* P, which specifies how each layer of a model M is transformed. Applying P yields the compressed

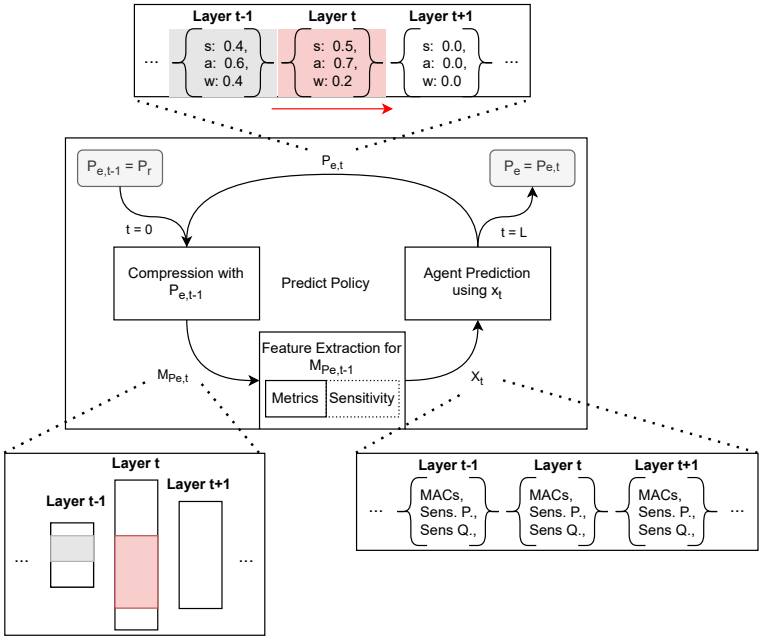


Figure 4.2 Intra-episode policy prediction cycle: at each step the agent proposes pruning/quantization parameters for one layer and appends them to the policy before proceeding. Reproduced with permission from [21].

model M^P. A policy consists of normalized compression method parameters (CMPs), where each parameter lies in the interval $[0,1]$:

$$P \in \{r \in \mathbb{R}^K \mid r_i \in [0,1]\}^{L \times M}, \tag{4.1}$$

with L the number of layers, M the number of supported methods, and K the dimensionality of parameters per method. Although pruning and quantization are discrete in nature (channels, datatypes, bit widths), CMPs are expressed as continuous variables to provide a uniform representation across layers. A mapping step later converts these normalized values into discrete hardware-specific configurations. This design reduces the complexity of the search space: the reinforcement agent always operates on normalized CMPs, without first having to learn layer-specific channel counts or quantization ranges.

The search problem is formulated as a constrained optimization task:

$$\hat{P} = \arg\max_{P} \; acc(M^P(\theta; x), y) \quad \text{s.t.} \quad cost(M^P) \leq c \cdot cost(M), \tag{4.2}$$

where $acc(\cdot)$ is the accuracy of M^P on data (x, y), θ the model parameters, and c the target cost ratio. Unlike prior works that approximate cost with

FLOPs, parameter counts, or BOPs [35], [72], [103], *Galen* instead uses measured inference latency on the target device as the actual optimization constraint [21].

4.2.2 Algorithmic Schema

As shown in Figures 4.1 and 4.2, an *episode* corresponds to compressing a model once. At each step t, the agent observes the current layer features s_t (layer type, index, number of parameters, sensitivity score) and predicts continuous actions a_t, which encode pruning and quantization policies. The sequence of decisions across all layers forms a complete compression policy P_e. The model M^{P_e} is compiled with Apache TVM [115], executed on the target hardware, and evaluated for accuracy. Leveraging TVM enables *Galen* to target a wide range of CPU and GPU architectures. Retraining of compressed models for 1–5 epochs is included in each episode to recover accuracy. This step is particularly important for pruning, where performance initially drops substantially but typically recovers after only a few epochs. The measured latency and accuracy together define the reward $R(P_e)$ used to update the agent.

4.2.3 Compression Methods

Structured pruning Galen performs structured pruning by removing *output channels* from convolutional or linear layers, following an ℓ_1-norm criterion [149], [156]. Skip connections and other structural constraints are respected by the pruning procedure.

Quantization *Galen* supports three quantization schemes: mixed precision with independent bit widths between 1 and 8 (*MIX*), fixed-point 8-bit integer quantization (*INT8*), and no quantization (*FP32*). Mixed-precision support is hardware-dependent and not available for all hardware backends.

For each layer, the agent predicts continuous policy values $q_t^W, q_t^A \in [0,1]$ for weights and activations. These are first mapped to one of the three datatypes by a threshold rule:

$$d(q) = \begin{cases} \text{FP32}, & q \leq \tau_{\text{fp}}, \\ \text{INT8}, & \tau_{\text{fp}} < q \leq \tau_{\text{int}}, \\ \text{MIX}, & q > \tau_{\text{int}}. \end{cases} \tag{4.3}$$

If a layer is assigned FP32 or INT8, its tensors are kept in full precision or quantized to 8 bits, respectively. If the layer is assigned MIX, a bit-serial operator is used, and the agent's policy additionally determines the bit width $b \in \{1,\ldots,8\}$ for weights and activations. In this case, a real-valued input r is mapped to its quantized form as

$$Q(r) = \max(-n,\, \min(n,\, \lfloor s \cdot r - z \rfloor)), \tag{4.4}$$

with

$$n = 2^b - 1, \qquad s = \frac{n}{x_{\max} - x_{\min}}, \qquad z = \lfloor s \cdot x_{\min} \rfloor + 2^{b-1}. \tag{4.5}$$

where $x_{\min}$ and $x_{\max}$ denote the calibrated tensor range. Quantization uses uniform asymmetric quantization with dynamic range calibration. During search, we employ fake quantization to simulate reduced precision in training.

4.2.4 Sensitivity Analysis

To guide compression, Galen augments the agent state with per-layer sensitivity scores. These scores quantify how strongly a layer reacts to compression. For each candidate layer l, only that layer is compressed at a chosen strength, while all others remain uncompressed. The output distribution of the modified network is then compared with the baseline network using the Kullback–Leibler divergence [251], following the ZeroQ methodology [56]:

$$S(l) = \frac{1}{N} \sum_{j=1}^{N} D_{\mathrm{KL}}\Big(f(x_j) \,\|\, f^{(l,q)}(x_j)\Big), \tag{4.6}$$

where f is the original network, $f^{(l,q)}$ the network with only layer l compressed at strength q, and x_j calibration inputs. Low $S(l)$ indicates tolerance, high $S(l)$ indicates fragility. These scores are included in the state representation s_t, steering the agent toward hardware–robust trade-offs.

4.2.5 Direct Hardware Latency Benchmarks

Proxy cost models such as FLOPs, MACs, or BOPs do not capture real device behavior [72], [103]. Galen therefore benchmarks each compressed model directly on the target hardware. Compiled with TVM [115], each candidate M^P is executed to measure T^{M^P}, which is used as part of the reward.

4.2.6 Reward Function

The reward function follows the absolute formulation of Bender et al. [53], balancing accuracy and latency:

$$r(P) = acc(M^P) + \beta \cdot \left| \frac{T^{M^P}}{c \cdot T^M} - 1 \right|, \tag{4.7}$$

where $acc(M^P)$ is the accuracy of the compressed model, T^{M^P} and T^M are the measured latencies of compressed and original model, c is the target compression ratio, and the negativ cost exponent $\beta < 0$ penalizes exceeding the latency budget. Exponential reward functions [102] were also evaluated but were less stable in practice.

4.2.7 Compression Agents

All Galen agents build on the Deep Deterministic Policy Gradient (DDPG) algorithm [199], which supports continuous actions. DDPG combines an actor–critic architecture with experience replay. Actor outputs are perturbed by Ornstein–Uhlenbeck noise [254] for exploration. Both actor and critic are implemented as fully connected networks with two hidden layers (400 and 300 neurons, ReLU). Target networks use soft updates with $\tau = 10^{-3}$, and the replay buffer stores 10^6 transitions.

Three agent types were trained: (i) a pruning agent predicting p_t; (ii) a quantization agent predicting (q_t^W, q_t^A); and (iii) a joint agent predicting both. While sharing the same backbone and training setup, the agents differ in state encoding and action dimensionality.

4.3 Experimental Evaluation and Discussion

We evaluate Galen on ResNet-18 [179] trained on CIFAR-10 [224], deployed on a Raspberry Pi 4B with an ARM Cortex-A72 CPU. Models are compiled with Apache TVM [115], and candidate configurations are executed on the device to obtain measured inference latencies. During the search, each candidate model is briefly retrained for 1-5 epochs before validation, while the final compressed models are retrained for 30 epochs.

Table 4.1 Compressed model performance per agent with target compression ratio c. Reproduced with permission from [21].

Method	c	MACs	BOPs	Latency	Accuracy
Uncompressed		$4.75 \cdot 10^{10}$	$4.86 \cdot 10^{13}$	330 ms	93.0 %
Pruning Agent		$1.42 \cdot 10^{10}$	$1.45 \cdot 10^{13}$	98 ms	93.0 %
Quantization A.	0.3	$4.75 \cdot 10^{10}$	$8.23 \cdot 10^{11}$	98 ms	92.5 %
Joint Agent		$4.35 \cdot 10^{10}$	$9.42 \cdot 10^{11}$	99 ms	93.2 %
Pruning Agent		$9.24 \cdot 10^{9}$	$9.45 \cdot 10^{12}$	66 ms	92.4 %
Quantization A.	0.2	$4.75 \cdot 10^{10}$	$4.01 \cdot 10^{11}$	57 ms	45.0 %
Joint Agent		$2.82 \cdot 10^{10}$	$6.74 \cdot 10^{11}$	64 ms	92.8 %

The quantization agent is trained for 310 episodes, while pruning and joint agents are trained for 410 episodes each, including 10 warm-up episodes. The reward function follows the absolute penalty form with a negativ cost exponent of $\beta = -3.0$, ensuring that policies converge close to the target compression budget.

General Performance Table 4.1 summarizes the quantitative results. At a moderate compression target (c=0.3), all agents achieve latency reduction close to the target and maintain accuracy near the baseline. At more aggressive compression (c=0.2), however, the quantization-only agent collapses to 45% accuracy because it is forced into extremely low bit widths. The pruning and joint agents, in contrast, remain accurate. As expected, pruning minimizes *MACs*, quantization reduces *BOPs* most efficiently, and the joint agent balances both—yielding the best accuracy–latency trade-off even though it is not maximal in either MAC or BOP reduction.

Figure 4.3 depicts the accuracy–latency trade-offs as the target compression ratio c varies. As expected, the pruning agent consistently achieves the largest reductions in MACs and only influences BOPs indirectly via structural changes, whereas the quantization agent is most effective at reducing BOPs. The joint agent consistently lands closer to the Pareto front: it is *not* the extreme winner on MACs or BOPs alone, yet achieves the best latency–accuracy trade-off for a given target latency by effectively balancing pruning and quantization.

Policy Analysis Figure 4.4 illustrates the layer-wise policies for c=0.3. The pruning agent spreads sparsity across most layers, with a tendency to prune later layers more strongly. The quantization agent varies bit widths across layers, typically quantizing weights more aggressively than activations—an observation

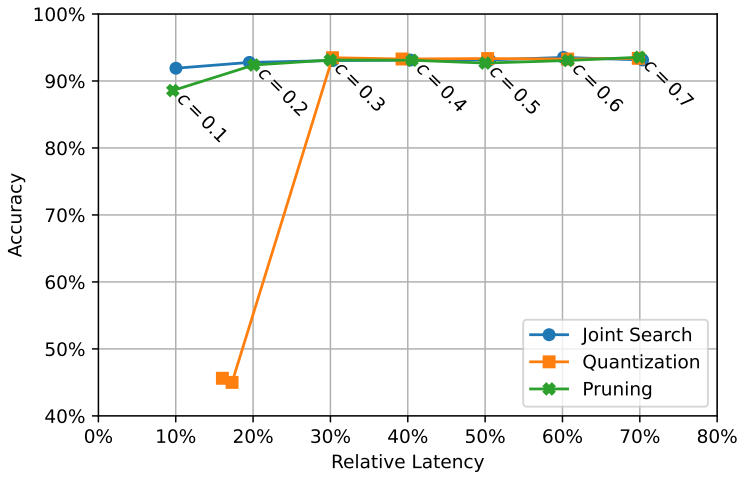
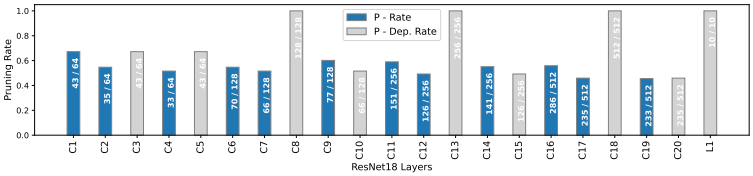


Figure 4.3 Accuracy–latency trade-offs across target compression ratios c: all three agents perform well for moderate c; under aggressive targets the quantization agent suffers accuracy loss from very low bit widths, while pruning and the joint agent remain competitive, with the joint agent slightly superior at the extreme. Reproduced with permission from [21].

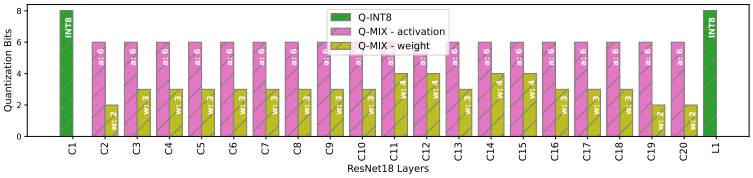
in line with expert knowledge for image classification models. As in common practice, the first and last layers are quantized less aggressively. The joint agent combines both methods in a balanced manner, moderating pruning and quantization simultaneously to preserve accuracy while meeting latency.

Sequential versus Joint Search To test the importance of joint optimization, we also evaluate sequential application of pruning and quantization. As Figure 4.5 shows, sequential pipelines tend to overuse the second stage, producing harsher compression and more accuracy loss. Joint search instead balances both simultaneously, yielding smoother policies and higher accuracy. This confirms that joint optimization is essential for Galen's effectiveness.

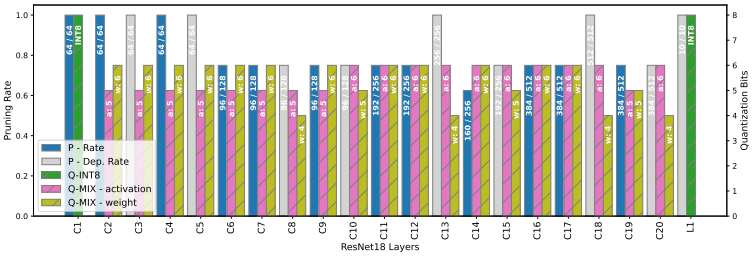
Role of Sensitivity Information Finally, we test whether Galen benefits from sensitivity features. Figure 4.6 shows that sensitivity varies across layers: later layers are especially fragile to quantization, while pruning sensitivity exhibits distinct heterogeneity probably relating to the skip connections of residual architectures. Table 4.2 and Figure 4.7 compare policies with and without sensitivity input. Without sensitivity, the agent produces almost uniform policies with nearly no heterogeneous structure. With sensitivity enabled, policies adapt to layer robustness, compressing tolerant layers more aggressively and protecting



(a) Pruning agent

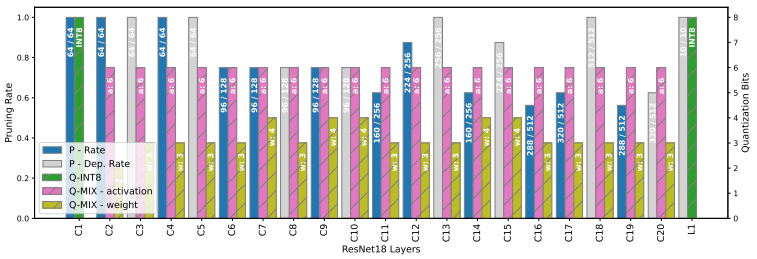


(b) Quantization agent

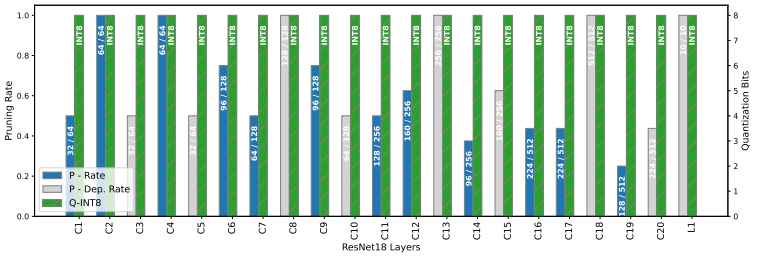


(c) Joint agent

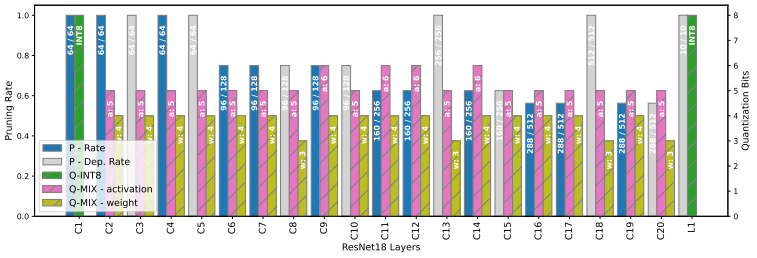
Figure 4.4 Predicted per-layer policies at $c = 0.3$ for pruning, quantization, and joint agents; bars show remaining channels (pruning) and activation/weight bit widths (quantization). Reproduced with permission from [21].



(a) Pruning → quantization

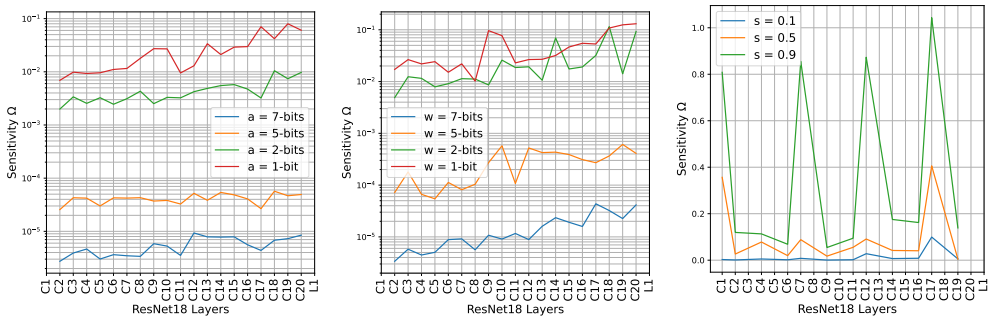


(b) Quantization → pruning



(c) Joint search

Figure 4.5 Sequential vs. joint search at $c = 0.2$: sequential pipelines overuse the second stage (harsher compression), whereas joint search balances pruning and quantization across layers. Reproduced with permission from [21].



(a) Activation sensitivity (b) Weight sensitivity (c) Pruning sensitivity

Figure 4.6 Per-layer sensitivity to activation quantization, weight quantization, and pruning. Reproduced with permission from [21].

Table 4.2 Quantitative results of joint search with sensitivity enabled vs. disabled at $c = 0.2$. Reproduced with permission from [21].

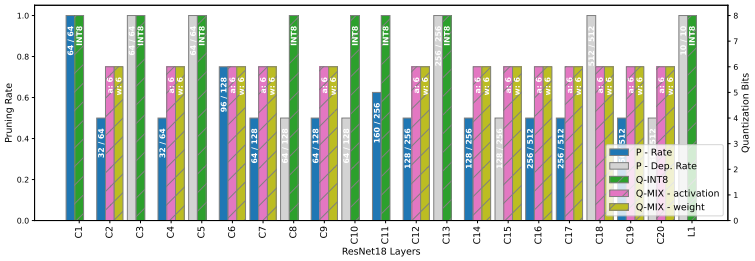
	MACs	**BOPs**	**Rel. Lat.**	**Accuracy**
Reference	$4.75 \cdot 10^{10}$	$4.86 \cdot 10^{13}$	100.0 %	93.04 %
Disabled	$1.68 \cdot 10^{10}$	$8.10 \cdot 10^{11}$	20.0 %	92.66 %
Enabled	$2.82 \cdot 10^{10}$	$6.75 \cdot 10^{11}$	19.4 %	92.77 %

fragile ones. This demonstrates that sensitivity information is essential for Galen to exploit per-layer heterogeneity effectively.

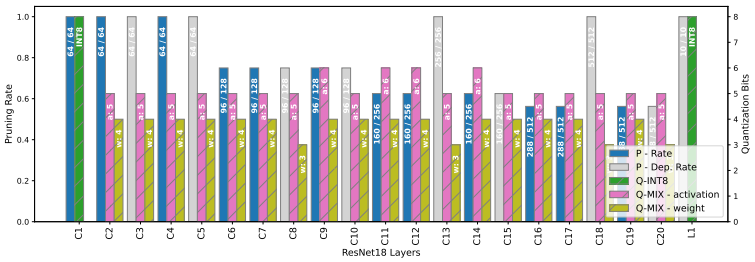
Summary

In this chapter, we introduced *Galen*, a framework for automatic, hardware-aware model compression. Galen learns per-layer compression policies directly on the deployment device, aligning the optimization objective with real execution behavior and bridging the gap between algorithmic design and hardware realization.

A central component of the framework is its *hardware-in-the-loop* approach, in which candidate models are compiled and benchmarked on the target system. This ensures that optimization is guided by measured latency rather than by analytical proxies, capturing real compiler, memory, and parallelization effects. Through *joint compression*, a single agent coordinates structural and precision reductions, yielding consistently favorable accuracy–latency trade-offs. Incorporating layer-wise *sensitivity information* further allows the agent to adapt compression strength to architectural heterogeneity, avoiding uniform and suboptimal policies. Finally, Galen demonstrates *expert-consistent behavior*: it rediscovers established heuristics such as conservative treatment of early and



(a) Without sensitivity



(b) With sensitivity

Figure 4.7 Joint policies at $c = 0.2$ without (a) and with (b) sensitivity features; sensitivity enables exploitation of per-layer heterogeneity. Reproduced with permission from [21].

late layers, stronger pruning in deeper stages, and higher tolerance of weights to quantization. This alignment between learned and human-designed strategies highlights both the interpretability and the robustness of the method, underscoring Galen's role as a step toward automated, trustworthy, and hardware-grounded model compression.

5

Modeling Analog Hardware Accelerators

Analog is more beautiful than digital, really,
but we go for comfort.

— Anton Corbijn

The previous chapter demonstrated how model compression can markedly reduce the computational footprint of neural networks by pruning redundant parameters and lowering numerical precision, while simultaneously accounting for hardware-specific constraints. Compression addresses one side of the efficiency spectrum: reducing the model's size and computational demand to facilitate execution on digital processors.

An orthogonal route to efficiency is to reduce the cost of computation itself by changing the underlying computing paradigm and performing inference directly in the physical domain, for example using analog or mixed-signal hardware accelerators. Rather than compressing the model, this approach makes the *computations* themselves cheaper by exploiting physical quantities such as currents, voltages, or charges to represent and manipulate numerical values. Such devices promise orders-of-magnitude gains in energy efficiency and compute density [46], [49],

yet they introduce new challenges arising from device mismatch, nonlinearities, saturation effects, and intrinsic noise [4], [44], [80], [99].

In the following chapter, we therefore turn from digital compression to analog hardware modelling, building on our publications [4], [44] and recounting how progressively refined models helped us both accelerate training for the BrainScaleS-2 (BSS-2) [49] system and uncover unexpected properties such as the non-associativity of analog dot products.

5.1 Analog Computing

Analog computing lacks the "safety net" of discretization and is therefore directly exposed to imperfections that can fundamentally alter the outcome of computations. Among the most critical are nonlinearities, saturation effects, dynamic phenomena such as leakage or crosstalk between circuit components, and various sources of noise, all of which can interact in complex ways and degrade accuracy [44], [49]. Analog accelerators are further affected by static device mismatch and variability across hardware instances due to manufacturing tolerances, as well as increased sensitivity to environmental conditions such as supply fluctuations and temperature [46], [135].

Naïve deployment of digitally trained models onto such accelerators can severely degrade predictive quality, in some cases reducing performance to the level of random guessing and thereby destroying all practical utility [44].

Nevertheless, artificial neural networks exhibit a remarkable tolerance to computational uncertainties when training explicitly accounts for them. This is well established in the context of digital quantization and pruning, both of which can be viewed as unsafe optimizations that introduce noise into the computation, yet typically retain accuracy after retraining [94], [190]. Building on this principle, several methods have been proposed to enhance robustness for analog accelerators, for example through knowledge distillation [80], explicit noise-injection during training [131], [236], or quantization-aware training [99]. These results demonstrate that, given appropriate training strategies, neural networks can adapt to significant levels of hardware-induced imprecision.

The most accurate way to expose a model to the full set of imperfections of a concrete analog chip instance is to include the hardware directly in the training loop. Such hardware-in-the-loop (HIL) training lets learning absorb

device-specific variations and dynamic effects, and has been shown to be highly effective on the BSS-2 neuromorphic system [36], [49].

Yet hardware-in-the-loop training suffers from practical limitations: dependence on physical device access, retraining overhead for each hardware instance, and limited throughput on some accelerators such as BSS-2, which slows down training compared to the scalability of digital simulation on conventional compute clusters. This motivates software models of analog accelerators that can be inserted into the training loop [44].

In this chapter we recount our path on BSS-2: we first built a white-box model that encodes measured nonidealities via simple, parallelizable components and demonstrated substantial training speedups and accuracy gains over simplistic baselines [44]. Persisting discrepancies led us to suspect temporal and ordering effects. We therefore tried an intentionally over-dimensioned transformer-set model to test whether sequence dependence mattered. That experiment revealed an unexpected and fundamental property: effective non-associativity of analog dot products on BSS-2 [4]. Both models extract the hardware behavior from measured data, using these measurements to characterize and encode the specific inaccuracies of the concrete device instance [4], [44]. Taken together, these models illustrate a central principle of our work: the value of hardware modelling lies not only in reducing the cost of training, but also in revealing which device characteristics fundamentally shape computation.

5.1.1 Related Work

The robustness of neural networks to imprecise computation has long been investigated in the context of quantization and pruning [94], [190]. Noise injection and knowledge distillation have been proposed to further enhance robustness against analog-style perturbations [80], [99], [166], [236]. For analog accelerators in particular, early modeling approaches often approximate imperfections as additive Gaussian noise or uniform quantization. While such abstractions facilitate hardware-in-the-loop training, they overlook device heterogeneity and structured nonlinear effects.

In summary, prior work has demonstrated the promise of analog accelerators [46], [49], [64], [124], [164] and has explored robustness through quantization, noise injection, and distillation [80], [99], [166], [190], [236]. Yet most models

remain either too simplistic—ignoring heterogeneity and dynamic nonlinearities—or too domain-general to capture device-specific effects. Our contribution is twofold: (i) a compact white-box model that leverages device-characterized data to encode static variations and stochastic noise, thereby bridging much of the gap between digital pretraining and hardware-in-the-loop retraining [44]; and (ii) a transformer-set model that revealed effective non-associativity of dot products in analog accelerators [4]. Together, these models highlight a guiding principle: software models of analog hardware are most valuable when they not only replicate hardware behavior but also *reveal* which imperfections are consequential for training and deployment.

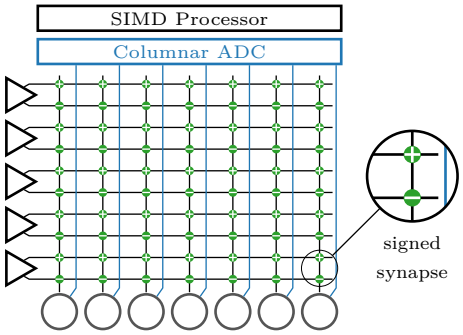
5.2 BrainScaleS-2

BSS-2 is a mixed-signal neuromorphic SoC manufactured in 65 nm CMOS technology, designed to support both the accelerated emulation of spiking neural networks and the efficient execution of analog matrix–vector products within its analog core [36], [49].

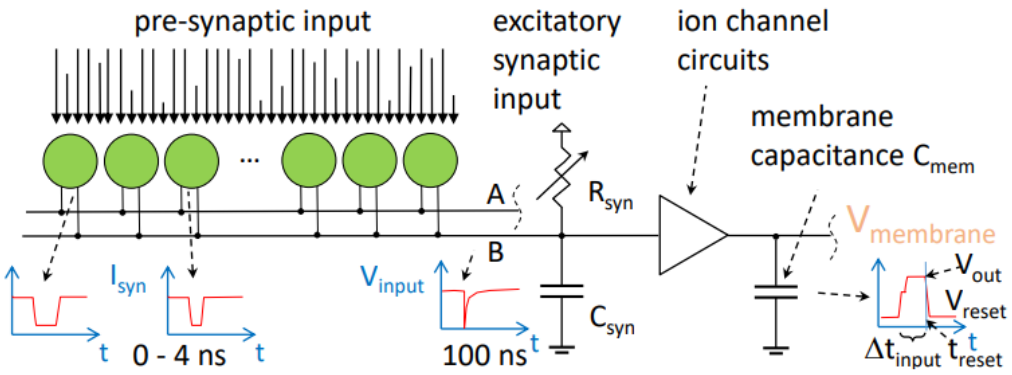
As illustrated in Figure 5.1(a), the core implements a crossbar array comprising 512 columns—corresponding to neuron circuits—and 128 rows that represent the input activations. To support signed weights, each logical column is realized by a pair of physical columns (256 virtual neurons in total), and each column is equipped with a dedicated per-column analog-to-digital converter (ADC) for digitized readout.

During a matrix–vector multiplication, each input activation a_i is encoded as a pulse length Δt_i (5 bit precision), while the synaptic weight $w_{i,j}$ is implemented as a current amplitude $I_{i,j}$. Each synapse thereby generates a current pulse with charge $Q_{i,j} = I_{i,j}\Delta t_i$, which is accumulated on the column capacitor of its target neuron via operational transconductance amplifier (OTA) circuits. The resulting membrane potential is then digitized by the column's 8 bit ADC. This integration of current pulses is illustrated in Figure 5.1(b).

This architecture, together with its characteristic nonidealities, makes BSS-2 a representative example for studying both the benefits and the limitations of analog matrix–multiply accelerators.



(a) Block diagram of the BSS-2 analog core, showing synapse drivers (triangles), synapses (small circles in the matrix), and neurons (large circles at the column ends). Reproduced with permission from [36].



(b) Detailed view of a single BSS-2 column in non-spiking mode, illustrating how synaptic pulses are integrated onto the neuron membrane. Reproduced with permission from [49].

Figure 5.1 Architecture of the BSS-2 analog neuromorphic system. Panel (a) shows the overall crossbar structure of the analog core, while panel (b) highlights the electrical details of a single column.

Experimental Setup

Dataset. All experiments in this chapter are based on the Google Speech Commands (GSC) dataset [137], a widely adopted benchmark for keyword spotting. It consists of one-second audio snippets of spoken keywords drawn from a vocabulary of 35 classes. We followed the standard split into training, validation, and test sets as provided in the dataset release.

Preprocessing. Input waveforms were converted into log-Mel spectrogram features. This representation is well suited for keyword spotting and, importantly, produces non-negative values only. The latter property aligns with the constraints of the BSS-2 hardware, which supports only positive activations.

Model architecture. The same network architecture was used for all experiments with both the white-box and transformer-set models (Fig. 5.2). It is designed to balance task performance with hardware interpretability: large enough to solve the keyword spotting task, yet compact enough to make hardware effects transparent. The first layer is considerably wider than subsequent layers and therefore cannot be mapped onto the hardware in a single call; instead it is split across multiple sequential calls. All following layers fit directly on the hardware without partitioning. This design respects hardware constraints while keeping the analysis of imperfections tractable.

5.3 White-Box Model of BSS-2

Hardware-in-the-loop retraining is the most accurate way to adapt neural networks to analog accelerators, since it directly exposes the model to the device's imperfections. However, it is time-consuming and scales poorly when device throughput and access are limited. As an alternative, we investigated whether a compact and interpretable software model of hardware nonidealities can replace most hardware-in-the-loop training epochs without compromising accuracy. In the following, we present the design of this white-box model and summarize its key findings; a more detailed treatment can be found in [44].

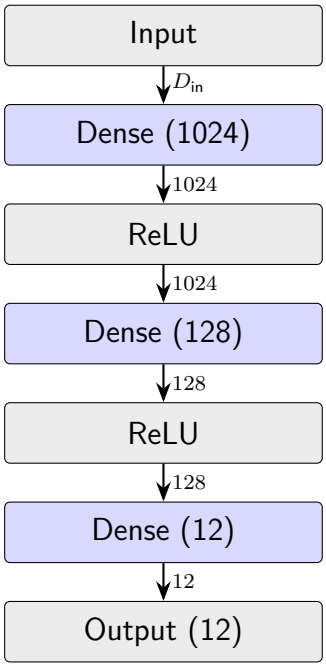


Figure 5.2 Neural network architecture used for keyword spotting on the Google Speech Commands [137] dataset. A 3-layer MLP with hidden sizes 1024 and 128 and a 12-way output; activation dimensions are annotated on the arrows. The first (wide) layer is mapped to hardware in multiple sequential calls, while subsequent layers map directly. Reproduced with permission from [44].

5.3.1 Model Description

The white-box model explicitly encodes the main sources of imperfections observed on BSS-2. Specifically, it combines lookup tables for per-synapse nonlinearities, spline functions that capture column-specific saturation effects of the membrane integrators, and Gaussian noise terms modeling stochastic fluctuations. Figure 5.3 illustrates this schematically, highlighting their components and their relation to the underlying hardware components. Each component was parameterized from direct hardware measurements. Together, these components provide a structured yet lightweight representation of the device's non-ideal behavior.

Lookup table for synaptic multipliers. To capture static local nonidealities at the level of individual synapses, the model uses lookup tables (LUTs) that encode their nonlinear characteristics. Each hardware synapse is intended to realize a multiplication between an activation a_i (encoded as a pulse length Δt_i) and a weight $w_{i,j}$ (encoded as a current amplitude $I_{i,j}$). In practice, unavoidable manufacturing variations give rise to device mismatch, which in turn

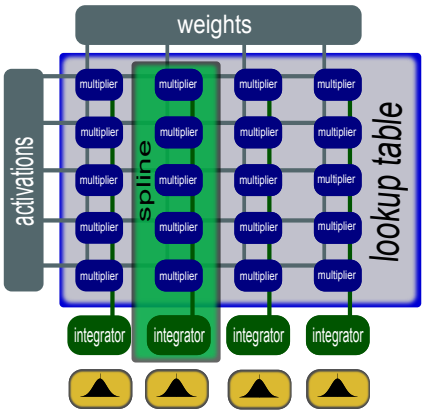


Figure 5.3 Schematic overview of the white-box representation: *lookup table* for the static variations of the multiplier array, *splines* for the column and integrator specific variations, and additive *Gaussian noise* that models the electrical noise, among others. Reproduced with permission from [44].

causes deviations from the ideal behavior, resulting in nonlinear and synapse-specific responses. We characterized these effects by measuring row-wise transfer functions, activating one synapse at a time with controlled inputs so that column saturation did not interfere with the measurement. The resulting input–output curves were stored as lookup tables $e_{\text{lookup}}[r, c, a, w]$, which constitute the first stage of the white-box model and provide a calibrated mapping from operand values to synaptic contributions.

Column-wise splines for saturation effects. When multiple synapses in a column are active, their contributions interact nonlinearly due to line saturation and OTA limitations. To model this, the summed LUT outputs of a column are passed through a column-specific spline,

$$\tilde{y}_c = f_c\left(\sum_r e_{\text{lookup}}[r, c, a_{rc}, w_{rc}]\right). \tag{5.1}$$

These splines were fitted to measurements taken under full-range excitations (activating many synapses at once), capturing the column's characteristic saturation curve. The spline stage thus models column-specific interaction effects beyond isolated multipliers.

Gaussian noise with per-neuron variance. Electrical noise further perturbs outputs and is not constant across neurons and varies with the input vector

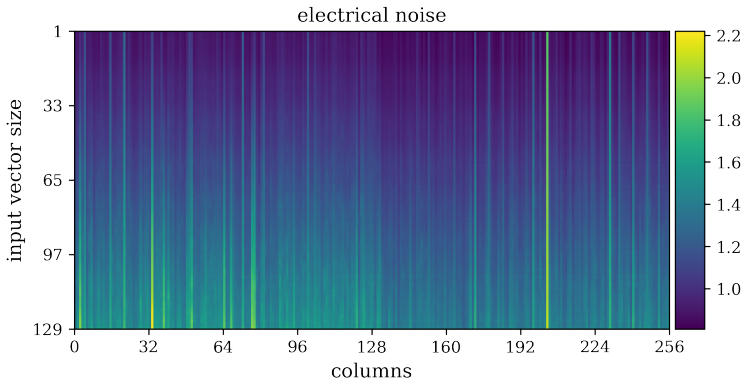


Figure 5.4 Electrical noise measured as the standard deviation of repeated calculations with identical inputs. With larger input vectors the average noise increases (vertical gradient), since more noisy values are accumulated. At the same time, variances between columns are significant, as shown by the vertical spread. Reproduced with permission from [44].

lengths. We estimated noise variances by repeatedly measuring identical inputs and recording output variability. This yielded a variance profile $\sigma^2(\text{col}, N)$ depending on the column index and the number of active inputs N, as illustrated in Figure 5.4. At runtime, we add this calibrated zero-mean Gaussian noise to each neuron output, reproducing the measured stochastic behavior,

$$y_c = \tilde{y}_c + \mathcal{N}(0, \sigma_c^2). \tag{5.2}$$

Incremental noise training. A further improvement was achieved through incremental noise training. Here, the Gaussian noise level was gradually increased from 0% to 100% of the measured variance over the course of training, following a curriculum strategy. This gradual exposure improved robustness by allowing the network to adapt progressively to hardware imperfections, and its effect is clearly visible in Table 5.1.

5.3.2 White-box Model Results

Figure 5.5 illustrates the progressive effect of the white-box corrections. On the left, raw hardware results deviate strongly from the ground-truth matrix multiplication. Applying the lookup table corrections (middle) compensates for much of this error, accounting for synapse-specific nonlinearities. Adding the

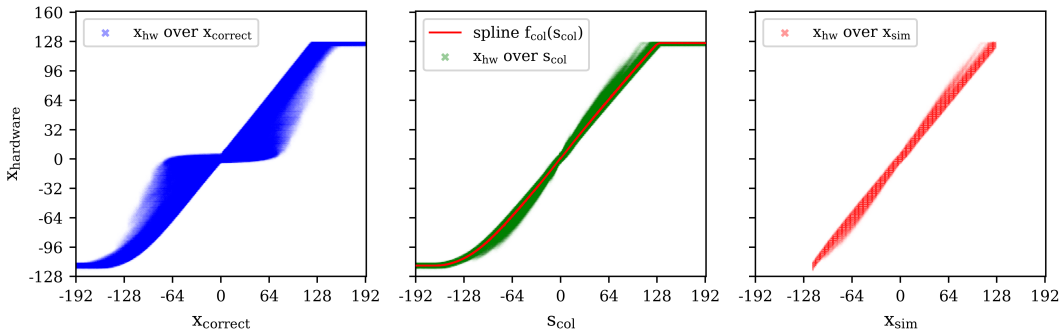


Figure 5.5 Mismatch between the ideal matrix multiplication result x_{correct} and the measured hardware output x_{hardware} for a random column (left). A large fraction of this error can be compensated by applying the lookup table sums s_{col} (middle) together with a spline interpolation that captures column-specific saturation effects (right). An ideal representation would follow the identity relation, broadened only by electrical noise. Reproduced with permission from [44].

spline corrections (right), which represent column-level saturation effects, brings the response close to the ideal identity relation. The resulting curve is very close to the ideal identity relation; the remaining deviations are attributable to residual effects not captured by the model, and ultimately to stochastic noise.

To quantify how closely different representations approximate hardware behavior, we compared the error of several modeling approaches across input sizes (Fig. 5.6). A simple linear regression model performs worst, as it cannot capture either local nonlinearities or saturation. Columnar approximations improve on this by accounting for average saturation behavior, but remain limited. Our proposed white-box model consistently performs best, closely tracking the hardware outputs across input sizes. Nonetheless, a residual gap to the hardware remains, indicating that not all effects are captured by the white-box model. At this stage we could only speculate that additional phenomena—such as temporal or ordering effects—might play a role. This hypothesis motivated the development of a more expressive transformer-set model (Sec. 5.4).

As summarized in Table 5.1, these corrections translate into substantial accuracy gains in end-to-end training. The white-box model consistently outperformed the quantization baseline and narrowed the gap to full hardware-in-the-loop training. When combined with a small number of final hardware-in-the-loop epochs, it even surpassed prolonged pure-hardware-in-the-loop training while reducing retraining time from more than 10 h to under 2 h. This demonstrates that

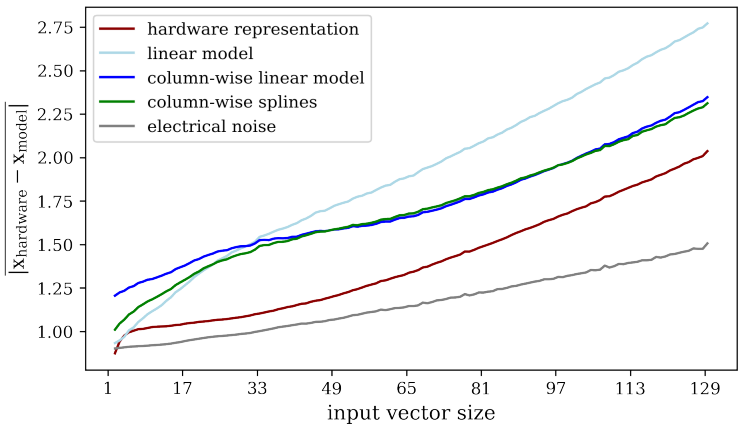


Figure 5.6 Model performance measured as the deviation from hardware results on random inputs. An ideal model would align with the electrical noise (gray) observed on hardware. The proposed hardware representation outperforms linear-regression and spline-based models. Both noise and model imperfections increase with input vector size due to more involved components and saturation effects. Reproduced with permission from [44].

device-characterized models can both accelerate training and improve accuracy compared to simplistic approximations.

The white-box model highlights that explicitly encoding static nonlinearities and noise structure is crucial for analog accelerator training. Its main advantages are efficiency, parallelizability, and interpretability: each component maps directly to a physical imperfection. However, residual discrepancies remained, especially under temporally clustered high inputs. Because the LUT + spline + noise model is fundamentally static, it cannot capture ordering- or time-dependent effects. This limitation motivated our next step: testing a more expressive transformer-set model, which ultimately revealed the phenomenon of effective non-associativity.

5.4 Transformer-Set Model and Non-Associativity

5.4.1 Hypothesis: Could Ordering Matter?

Despite the progress of the white-box model, a residual gap to the hardware remained (Sec. 5.3), which could not be explained by static nonlinearities, saturation, or stochastic noise alone. At this stage we speculated that the order in which synaptic pulses arrive might influence the computation. If true, this would

Table 5.1 Comparison of retraining strategies. Reproduced with permission from [44].

Method		Retraining		Accuracy	
Name	**Description**	**Epochs**	**Time [min]**	**SW**	**BSS-2**
Plain	Full-precision baseline	0	0.0	80.8 %	12.3 %
Quantized	INT6 weights, UINT5 activations	300	13.1	79.6 %	25.8 %
Noise only	Baseline with Gaussian noise	300	10.4	80.5 %	18.4 %
Rep. no noise	Static variances, no noise	300	83.4	76.5 %	26.4 %
Rep. with noise	Hardware representation with noise	300	83.4	73.5 %	35.1 %
Rep. incr. noise	Hardware representation, incr. noise	300	83.6	76.0 %	41.0 %
HIL (300)	Full hardware-in-the-loop	300	652.3		66.8 %
HIL (350)	Full hardware-in-the-loop	350	769.5		66.7 %
Comb. quant. (5)	Quantization (300) + HIL (5)	305	13.1 + 11.5		62.1 %
Comb. quant. (50)	Quantization (300) + HIL (50)	350	13.1 + 117.5		67.3 %
Comb. rep. (1)	Representation (300) + HIL (1)	301	83.6 + 2.2		64.9 %
Comb. rep. (5)	Representation (300) + HIL (5)	305	83.6 + 11.5		67.4 %
Comb. rep. (10)	Representation (300) + HIL (10)	310	83.6 + 23.5		69.7 %
Comb. rep. (50)	Representation (300) + HIL (50)	350	83.6 + 117.5		**70.1 %**

violate the assumption of associativity: mathematically, addition is associative and commutative, $(x+y)+z = x+(y+z)$ and $x+y = y+x$, and therefore the dot product $\sum_i a_i w_i$ should be order-independent.

However, we hypothesized that analog dynamics such as line saturation or leakage could lead to order dependence. To test this hypothesis, we designed a deliberately expressive model with sufficient capacity to learn ordering effects if they exist.

5.4.2 Testing the Hypothesis with a Transformer Set

We constructed a set of Transformers [168], one independent Transformer per column (256 in total), trained to map input sequences (a_i, w_i) to the corresponding analog output (Fig. 5.7). Transformers were chosen because of their ability to capture sequence dependencies and thus provide an upper bound on what could be learned from ordering information. To isolate the role of order, we prepared two datasets:

- **Ordered:** operands presented in the true sequence as processed by the hardware.
- **Non-ordered:** operands permuted randomly but consistently across all rows, thereby preserving the multiset of inputs but destroying ordering information.

Both datasets were based on the same hardware-characterized measurements and included per-column Gaussian noise consistent with Sec. 5.3.

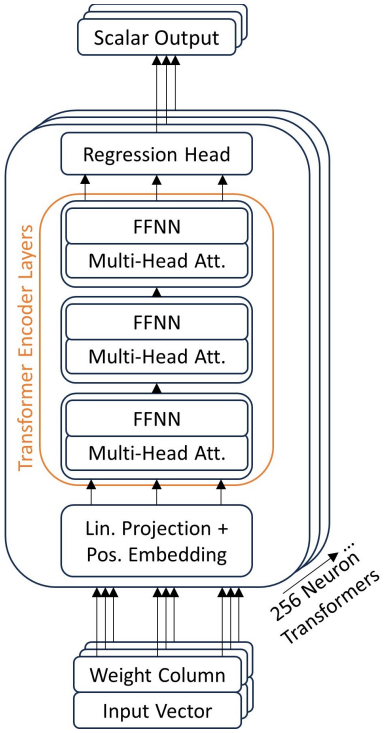


Figure 5.7 Transformer-set model: a deliberately expressive black-box model to probe for ordering effects. Reproduced with permission from [4].

5.4.3 Findings: The Emergence of Non-Associativity

Figure 5.8 illustrates an example of non-associativity: identical operands yield different results depending on their input order. Table 5.2 quantifies this effect: Google Speech Commands models trained with the ordered transformer-set representation fit the hardware significantly better (lower MSE) and achieved higher test accuracy after deployment on BSS-2 than models trained with either the non-ordered transformer set or the white-box model. The improvement,

Table 5.2 Comparison of hardware models. MSE to BSS-2 measurements and Google Speech Commands test accuracy when retrained with each model and deployed on hardware. The ordered transformer set performs clearly better, showing that operand order is a decisive factor. Reproduced with permission from [4].

Method	**MSE to BSS-2**	**GSC Test Acc.**	
		SW	**BSS-2**
Ordered Transformer Set	0.5	76.6%	53.4%
Non-ordered Transformer Set	0.9	77.8%	44.5%
White-box Model	-	76.9%	41.2%

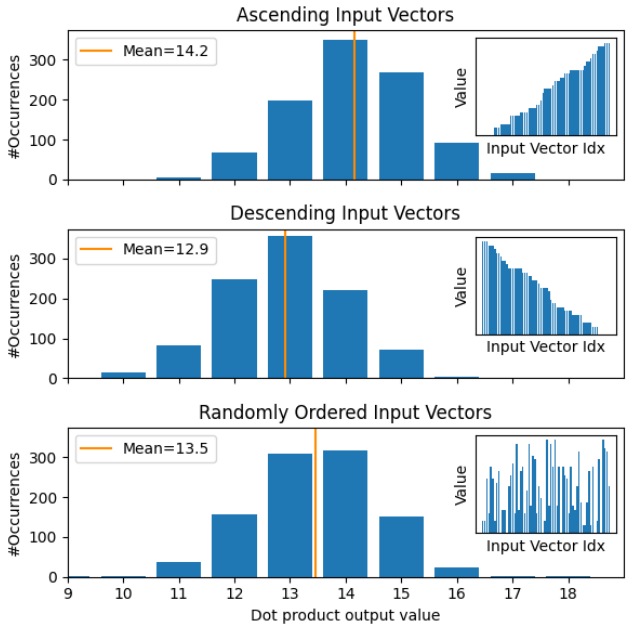


Figure 5.8 Example of non-associativity: identical operands produce different results depending on their order of arrival. Reproduced with permission from [4].

amounting to gains of up to 9–12% on keyword spotting, demonstrates that ordering is not a nuisance factor but a first-order effect of analog computation. Nonetheless, a smaller residual gap to pure hardware-in-the-loop retraining remained, indicating that additional time-dependent effects beyond operand ordering are also present.

Finally, we compared how the models behave when combined with a subsequent hardware-in-the-loop retraining phase (Table 5.3). Here, the transformer-set and the white-box model reached comparable accuracy, both reducing the required number of hardware-in-the-loop epochs by an order of magnitude compared to training without any hardware model. This highlights the complementary role of such models: they accelerate training and reveal which device effects matter, even if ultimate performance still requires direct hardware adaptation.

Summary

The two hardware models were developed for distinct purposes, yet together they have shaped how we train and reason about analog accelerators within this work. The white-box model was designed for interpretability and efficiency: it encodes static heterogeneity and stochastic noise in a compact form, runs efficiently on

Table 5.3 Google Speech Commands test accuracy after hardware-in-the-loop retraining. Using either the transformer set or the white-box model reduces the number of required HIL epochs dramatically compared to training without any model. Reproduced with permission from [4].

Method	HW Model Epochs	BSS-2 Epochs	Accuracy on BSS-2
Without HW Model	-	350	69.6%
Ordered Transformer Set	300	50	71.8%
Non-ordered Transformer Set	300	50	71.6%
White-box Model	300	50	71.9%

GPUs, and substantially reduces the need for hardware-in-the-loop retraining. The transformer-set model, by contrast, was deliberately over-expressive and served as a diagnostic tool. It revealed the surprising phenomenon of non-associativity due to ordering-dependent saturation—a property invisible to static models but essential for understanding analog computation.

From these insights a pragmatic recipe emerges: start with a conventionally trained digital model, apply quantization-aware training to obtain a suitable quantized baseline [94], [190], continue with white-box training (including incremental noise curricula) to capture device-specific variations, and conclude with a short hardware-in-the-loop fine-tuning. Where order effects are suspected to be significant, a targeted transformer probe can quantify their impact and guide both algorithmic and hardware-side mitigations.

Looking ahead, more detailed circuit-level models that explicitly represent hardware components and integrate their dynamics over time may be required to fully capture complex effects beyond ordering. Such physically faithful models can provide valuable insights, but their computational cost makes them impractical for use in the training loop of large-scale networks. Instead, lightweight representations such as the white-box model strike a balance between fidelity and usability. More broadly, hardware models serve a dual purpose: they inform circuit design and calibration strategies on the hardware side, while guiding neural architecture and training design on the algorithmic side. The guiding principle that has emerged from our work is clear: models should not aim to perfectly emulate hardware, but to provide understanding and enable the training of networks that are robust to its peculiarities.

6

Robustness Against Noisy Computations

What does not kill me makes me stronger.

— Friedrich Nietzsche, *Twilight of the Idols* (1888)

The preceding chapter addressed analog hardware accelerators from a device-centric perspective, focusing on how their nonidealities—such as nonlinearities, saturation, noise, and even ordering effects—can be captured in faithful models. These models serve as powerful tools for understanding the behavior of specific hardware instances and for enabling hardware-in-the-loop training. Yet, their strength lies precisely in their detail, and this level of fidelity also makes them less practical as a basis for generic robustness strategies on the algorithmic side.

This chapter therefore shifts the focus from *modeling hardware* to *hardening neural networks against noisy computations*. Rather than reproducing the peculiarities of one device instance, we take a broader perspective: How can robustness be measured, understood, and systematically improved such that neural networks remain reliable when deployed on analog accelerators? The emphasis thus moves to the machine learning side of the co-design problem.

To structure this discussion, we follow a trajectory across three works that

form the backbone of this chapter. We begin with *Walking Noise* [9], which extends our earlier workshop paper [16] and introduces a methodology to measure robustness at the granularity of individual layers, uncovering characteristic learning dynamics such as weight scaling and self-binarization. Building on these diagnostic insights, we then turn to *hardening methods* [7], where quantization-aware training and noisy training are systematically compared. This establishes noisy training as the baseline countermeasure, while clarifying the complementary role of quantization. Finally, we extend noisy training to *Variance-Aware Noisy Training (VANT)* [8], which addresses the fundamental limitation of static noise assumptions by introducing time-varying noise schedules, thereby improving robustness to practical effects such as drift and environmental variability.

Together, these contributions trace a coherent narrative: from *measuring robustness*, to *establishing effective countermeasures*, to *sustaining robustness under dynamic and realistic conditions*. This progression highlights how robustness is not a singular property but a layered challenge that requires both diagnostic tools and adaptive training methods.

This chapter builds on three of our publications—*Walking Noise* [9], *Hardening* [7], and *VANT* [8]—and focuses on their overarching narrative and distilled insights. Detailed methodologies, experimental setups, and quantitative results are provided in the respective original works.

6.1 Robustness Against Hardware Noise

The robustness of neural networks against noise has been studied from multiple perspectives, spanning generalization in machine learning, adversarial robustness, and hardware-induced uncertainty. A classical line of research established *noise injection* as a form of regularization to improve generalization, with early works comparing it to weight decay and early stopping [225], [233]. With the advent of adversarial attacks, noise injection was also investigated as a defense strategy, often in combination with adversarial training [191]. Variants include additive Gaussian perturbations to the inputs [235], ensembles with layer-wise noise injection [125], and differentiating diffusion- versus jump-term randomness in continuous-time neural stochastic differential equations [68].

A second line of work focuses on *noisy hardware*, where perturbations do not only affect the inputs but arise intrinsically from the computations themselves,

e.g., during weight readout or multiply-accumulate operations. Training with additive Gaussian noise has been shown to mitigate such hardware-induced degradation [84], [109], with further extensions such as Bayesian fine-tuning in *BayesFT* [78]. In resistive memories, perturbation models have captured effects such as drift in resistive random-access memory (RRAM), and training with injected noise has been shown to mitigate performance degradation [58].

To counteract accuracy degradation due to noisy computations, noisy training has also been successfully applied: while some works injected additive zero-mean Gaussian noise with varying variances during training [84], [109], *Noisy Machines* [80] extended this approach with knowledge distillation from a digitally trained teacher to a noisy student. Beyond proposing distillation, it also provided insights into the higher sensitivity of deeper architectures compared to wider ones, explained through a mutual information analysis of information loss across layers. This line of work established noisy training, in combination with auxiliary mechanisms such as distillation, as a fundamental tool for enhancing robustness on noisy analog accelerators.

Beyond Gaussian injection, *perturbation-based methods* have been studied to improve both generalization and robustness. Perturbing weights can guide SGD into flatter regions of the loss landscape, as formalized by Sharpness-Aware Minimization (SAM) [39]. By penalizing sharp minima, SAM improves generalization and offers a degree of robustness against input perturbations. Related studies investigated perturbations of both weights and inputs to strengthen adversarial robustness [63].

Quantization provides a complementary perspective. Originally motivated by efficiency on digital accelerators, quantization implicitly injects structured noise by reducing numerical precision. Its effect on robustness has been studied extensively: some works report non-monotonic behavior of adversarial robustness across bit-widths [62], while others show that moderate quantization can increase resilience with little accuracy loss [117]. Defensive quantization methods explicitly control the Lipschitz constant to prevent error amplification [92], and further studies found that complex DNNs can absorb severe quantization through retraining, whereas smaller networks degrade more substantially [203]. From the analog accelerator perspective, quantization is often indispensable due to limited DAC/ADC precision, motivating the use of quantization-aware training to preserve robustness under such constraints [122].

In summary, prior research established noise injection as a versatile tool for generalization and robustness. For analog accelerators, noisy training and quantization emerged as common countermeasures. Building on these insights, the following sections first examine per-layer robustness through *Walking Noise*, highlighting how neural networks adapt to tolerate noise when trained with noise injection. We then compare hardening strategies such as quantization and noisy training, quantifying their effectiveness for analog computations. Finally, motivated by observations on real analog hardware, noisy training is extended to account for dynamic variations in noise—such as those caused by temperature fluctuations—through VANT.

6.2 Walking Noise

Robustness of neural networks is usually quantified at the network level, for example as overall accuracy degradation under noise. Such aggregate measures, however, conceal that sensitivity to perturbations is highly heterogeneous across layers: early layers may amplify noise, intermediate layers can compensate through redundancy, while later layers often remain fragile. For analog accelerators this distinction is particularly relevant, as noise may arise at different points in the computation chain and its impact depends on where in the architecture it occurs.

Walking Noise [9] addresses this by probing robustness at the level of individual layers. By injecting noise sequentially layer by layer, it enables the identification of weak spots and provides insights into how networks adapt to tolerate noise. This fine-grained view complements global robustness measures and serves as a diagnostic basis for developing hardening methods.

6.2.1 Methodology

The key idea of *Walking Noise* [9] is to inject noise selectively at one layer at a time, while keeping the rest of the network unaffected. By "walking" the noise across layers, robustness can be probed in a fine-grained manner. Three types of perturbations are considered: additive Gaussian noise, multiplicative Gaussian noise, and combinations thereof.

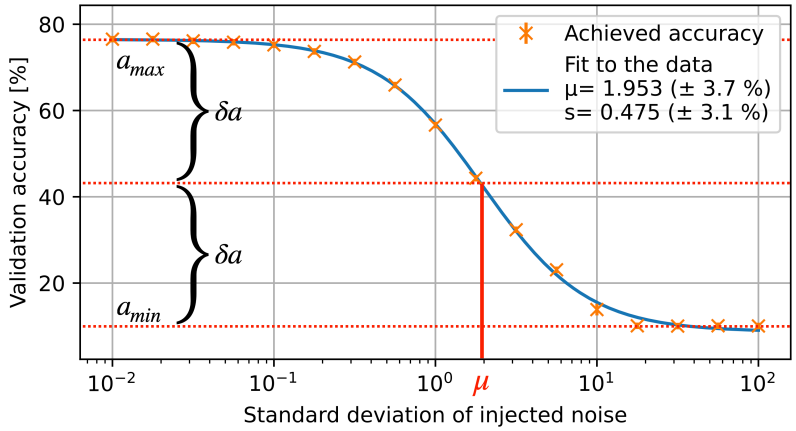


Figure 6.1 *Midpoint noise level* μ for the example of LeNet-5/CIFAR-10/BN and globally injected additive noise. Reproduced with permission from [9].

Depending on the accelerator, noise may arise at different stages of a matrix multiplication: (1) during weight readout, (2) in the multiply–accumulate itself, or (3) when forwarding activations. We focus on the last case, as it captures accumulated effects from both weight and computation noise.

To quantify robustness, the *midpoint noise level* μ is introduced. When increasing the globally injected noise, model accuracy typically follows a characteristic curve: it remains stable for low noise levels, then drops once a critical region is reached (Fig. 6.1). This behavior is well captured by fitting a logistic function to the accuracy–noise relation,

$$F(\sigma;\mu,s,\delta a,a_{min}) = \frac{2}{1+e^{(\sigma-\mu)/s}} \cdot \delta a + a_{min} \tag{6.1}$$

where $\delta a = (a_{\max} - a_{\min})/2$ denotes half of its maximum accuracy, σ the varying injected noise level, s the slope of the curve. The parameter μ of this fit denotes the standard deviation at which accuracy falls to the midpoint between its clean performance and random guessing. It thus provides a natural and comparable measure of robustness: the higher μ, the more noise the network or individual layer can tolerate before collapsing. Experiments cover standard convolutional networks on image classification benchmarks, trained both with and without noise injection, with details provided in [9].

6.2.2 Findings

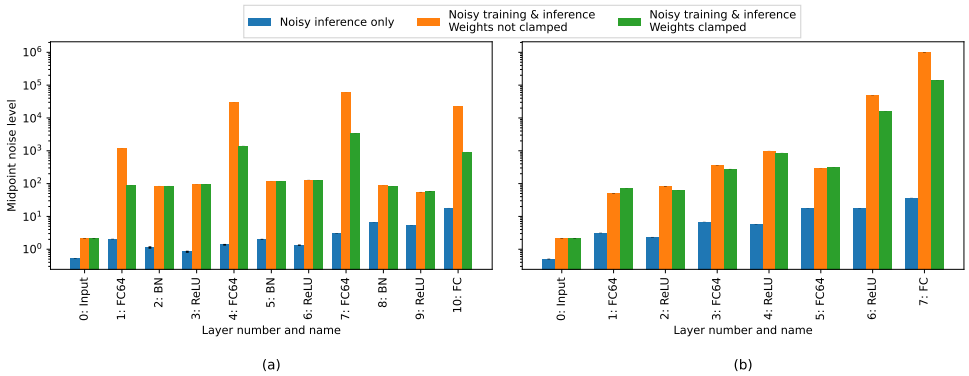
Applying the Walking Noise methodology reveals distinct robustness mechanisms depending on the type of perturbation and highlights strong heterogeneity across layers.

Additive noise. Under additive Gaussian perturbations, networks develop a compensatory mechanism by increasing the magnitude of their weights. This self-organized adaptation enhances the effective signal-to-noise ratio, allowing activations to remain separable despite the presence of noise. This behavior is clearly reflected in the characteristic accuracy–noise curves, which degrade smoothly with increasing perturbation strength before eventually collapsing to random guessing. The resulting robustness metric μ, derived from these curves, enables systematic comparison across datasets and architectures.

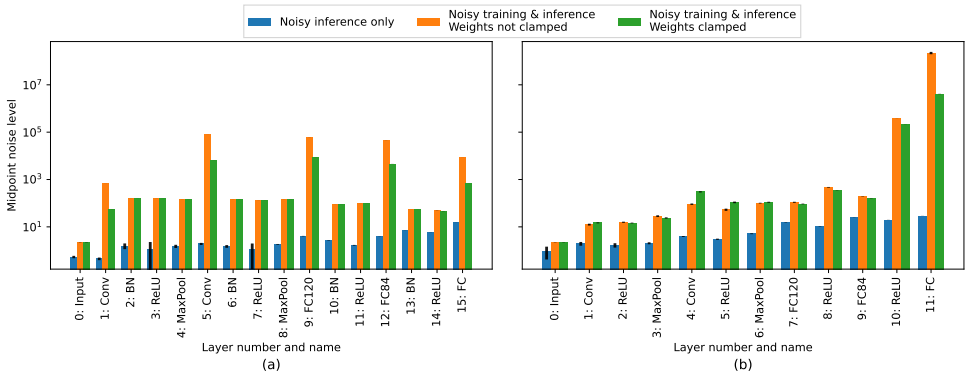
Batch normalization (BN) interacts strongly with this mechanism. Since BN normalizes activation scales after each layer, it counteracts weight growth and therefore suppresses the network's ability to exploit weight magnitude as a compensation strategy. As a result, models with BN often show lower robustness to additive noise than their non-BN counterparts (Fig. 6.2).

Multiplicative noise. In contrast to the additive case, multiplicative noise revealed an unexpected and remarkably strong form of robustness. When injecting multiplicative Gaussian perturbations during inference, we observed that some layers maintained high accuracy even for extremely large noise levels (up to $\sigma = 10^{10}$), without collapsing to random prediction accuracy. This behavior was particularly surprising, as no model could withstand such noise levels under additive injection or global perturbations.

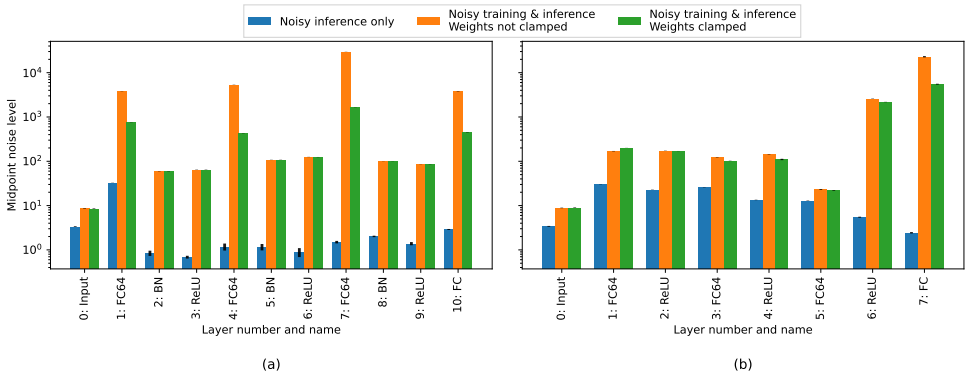
Closer inspection of the activation distributions showed that networks learn to *self-binarize*: activations split into two distinct peaks, one near zero and one near the noise standard deviation (Fig. 6.3). Information can then be encoded simply in the presence or absence of an activation peak, making the network insensitive to the precise values randomized within each cluster. The emergent binary representation explains the remarkable tolerance to multiplicative noise. This was evidenced by the fact that explicit threshold-based quantization without



(a) MLP on MNIST.



(b) LeNet-5 on MNIST.



(c) MLP on CIFAR-10.

Figure 6.2 Midpoint noise level for various model architectures and image classification datasets based on Walking Noise. The x axis shows layer number and name. Figures to the left are with BN, Figures to the right are without BN. Reproduced with permission from [9].

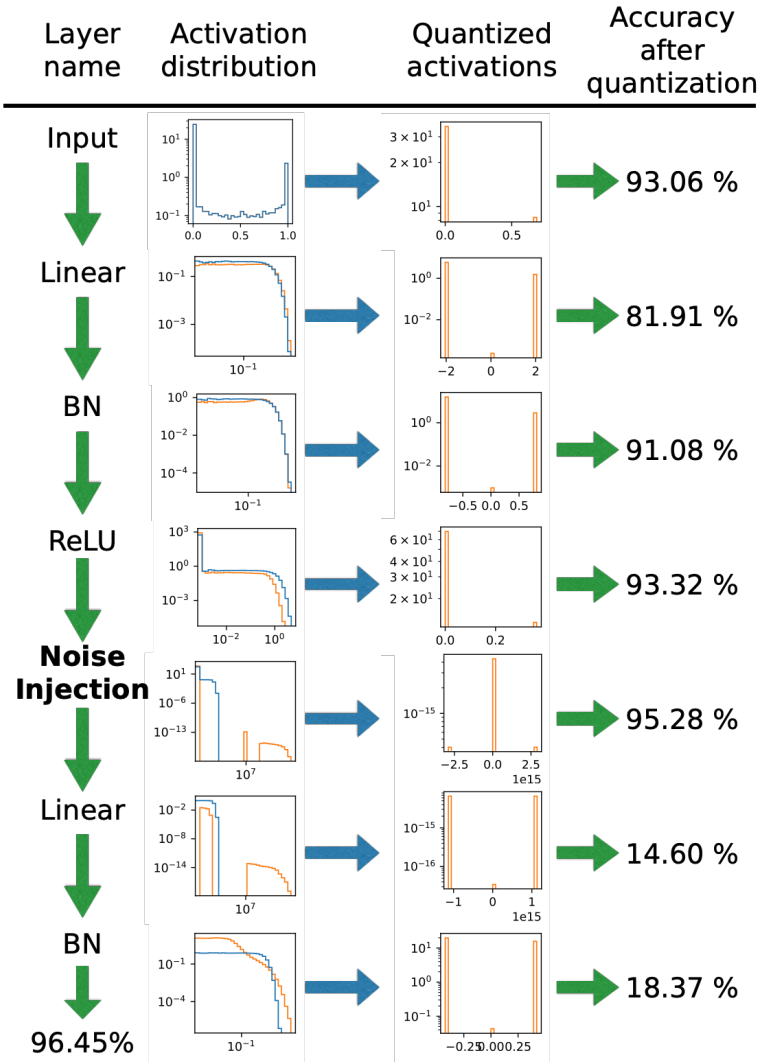


Figure 6.3 Visualization of *self-binarization* in an MLP trained on MNIST under multiplicative *Walking Noise*. The histograms on the left show the distribution of activation values with noise (orange) and without noise (blue) after layer execution, using both logarithmic and linear x-axis scales to emphasize differences between the two networks. On the right, the distribution of activation values is shown together with the resulting accuracy when applying simple threshold-based quantization without retraining to the respective layer. Reproduced with permission from [9].

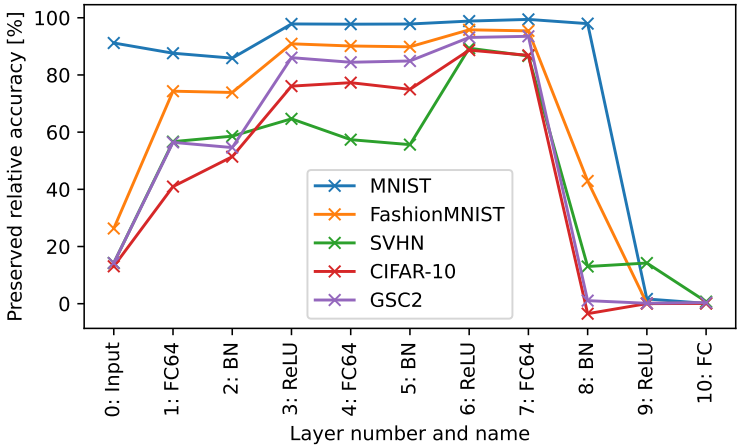


Figure 6.4 Accuracy preservation for MLP with BN and various datasets, noise injected multiplicatively. Reproduced with permission from [9].

retraining caused only a marginal accuracy drop, demonstrating that the network had already learned an internal binary representation.

Here, *batch normalization* is essential. By stabilizing the scale of activations, BN enables the formation and propagation of these bimodal distributions, and thus the self-binarization mechanism itself. Without BN, networks largely lose this ability, and accuracy drops quickly when multiplicative noise is injected. The effect generalizes across architectures and datasets (Figs. 6.4), though sensitivity again varies by layer, with later layers eventually failing to decode the binarized representation.

Mixed noise. Since the self-learned robustness strategies for additive and multiplicative perturbations differ fundamentally, their interaction in a mixed setting is of particular interest. Real-world analog matrix–multiply accelerators are subject to multiple sources of noise, and depending on the hardware, additive or multiplicative components may dominate and occur earlier in the computational chain. We therefore evaluate both injection orders:

$$\begin{aligned} \text{multiplicative-first: } & y(x) = \Big(x \cdot \mathcal{N}(1, \sigma^2_{\text{mul}})\Big) + \mathcal{N}(0, \sigma^2_{\text{add}}) \\ \text{additive-first: } & y(x) = \Big(x + \mathcal{N}(0, \sigma^2_{\text{add}})\Big) \cdot \mathcal{N}(1, \sigma^2_{\text{mul}}). \end{aligned} \tag{6.2}$$

Here, x denotes the clean input, while $\mathcal{N}(\mu, \sigma^2)$ represents Gaussian noise with mean μ and variance σ^2. The parameters σ_{mul} and σ_{add} control the standard deviations of the multiplicative and additive components, respectively.

Figure 6.5 illustrates the resulting accuracy surfaces under both injection

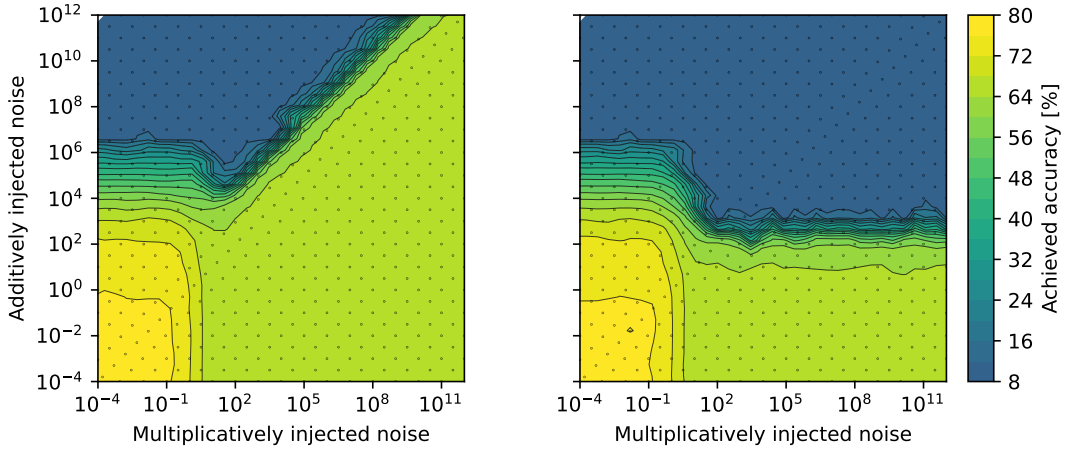


Figure 6.5 Accuracy for training with multiplicative and additive noise at layer 5 (Conv) of LeNet-5 trained on CIFAR-10. The black dots indicate points of measurement. Reproduced with permission from [9].

orders. A notable observation is that networks can still learn to self-binarize if multiplicative noise is applied first, thereby maintaining extreme robustness. This effect persists even when substantial additive noise is injected afterwards, in some cases up to one order of magnitude stronger. From a hardware perspective, this suggests a rather counterintuitive but powerful implication: in principle, even an accelerator dominated by strong additive noise could remain workable if its computation pipeline introduced multiplicative noise first. For practical accelerator design this means that hardware-in-the-loop trained networks may tolerate significant additive disturbances, provided these occur after multiplicative ones.

Layer-specific sensitivity. Walking Noise consistently demonstrates that robustness is highly heterogeneous across layers. By scanning noise layer by layer, weak spots of a network can be identified, often located in the first or last layers depending on the type of perturbation. This information can be exploited to guide targeted countermeasures. For example, selectively re-executing only the most sensitive layers yields significantly higher accuracy compared to uniformly repeating all layers (Table 6.1), thereby improving efficiency.

Summary. Overall, Walking Noise uncovers characteristic mechanisms of robustness in neural networks. For additive perturbations, robustness is achieved

Table 6.1 Walking Noise guiding multi-execution to improve accuracy. Reproduced with permission from [9].

			Executions per layer		Accuracy	
Dataset	**Model**	**BN**	**Uniform**	**Guided**	**Uniform**	**Guided**
MNIST	MLP	with	{2,2,2,...}	{6,1,3,3,2,1,2,1,1,1,1}	$71.4 \pm 0.7\,\%$	$\mathbf{80.6 \pm 0.5}\,\%$
MNIST	MLP	wo	{2,2,2,...}	{8,1,2,1,1,1,1,1}	$68.2 \pm 1.3\,\%$	$\mathbf{87.1 \pm 0.7}\,\%$
CIFAR-10	MLP	with	{2,2,2,...}	{1,1,3,4,2,2,3,2,1,2,1}	$41.7 \pm 0.5\,\%$	$\mathbf{43.7 \pm 0.5}\,\%$
CIFAR-10	MLP	wo	{2,2,2,...}	{4,1,1,1,1,1,2,5}	$38.9 \pm 0.6\,\%$	$\mathbf{43.4 \pm 0.7}\,\%$
CIFAR-10	LeNet-5	with	{2,2,2,...}	{3,2,3,3,3,1,3,4,2,1,1,2,1,1,1,1}	$58.5 \pm 1.4\,\%$	$\mathbf{61.1 \pm 1.0}\,\%$
CIFAR-10	LeNet-5	wo	{2,2,2,...}	{4,3,4,3,1,3,1,1,1,1,1}	$57.2 \pm 1.8\,\%$	$\mathbf{62.4 \pm 1.3}\,\%$

by increasing weight magnitudes, a strategy that is suppressed when *batch normalization* is applied, since batch normalization removes activation scale. For multiplicative perturbations, networks develop a self-binarization of activations, a mechanism that crucially depends on batch normalization to stabilize and propagate the bimodal distributions. Finally, under mixed noise conditions, the order of perturbation determines whether robustness can be preserved, with multiplicative-first injection enabling tolerance even in the presence of strong additive noise. Together, these findings establish robustness as a heterogeneous, layer-dependent property, and provide actionable insights for architecture design, compression strategies, and selective hardening.

Having established how robustness emerges at the layer level, we next investigate systematic strategies to *harden* neural networks against noise, comparing the effectiveness of quantization-aware training and noisy training.

6.3 Hardening Methods

The insights from Walking Noise highlight that robustness is not a uniform property, but instead emerges from heterogeneous layer-wise mechanisms. While this diagnostic perspective reveals how networks adapt to tolerate perturbations and helps to design more robust neural architectures, it leaves open the practical question of how to *systematically improve robustness* during training.

Two prominent methods are commonly considered. quantization-aware training (QAT) has long been used to prepare models for low-precision deployment, and implicitly introduces structured perturbations that may also enhance robustness. In contrast, noisy training directly injects stochastic perturbations during training, explicitly mimicking the conditions of noisy inference. Both approaches are widely applied, yet their relative effectiveness for analog computations—where

noise is unavoidable—had not been systematically compared.

This motivates the following study: to evaluate QAT and noisy training side by side across different architectures, and quantify their robustness using the midpoint noise level μ.

6.3.1 Methodology

To compare hardening strategies, we evaluate QAT and noisy training across several neural architectures, including LeNet-5, VGG-11, and ResNet-18, trained on the CIFAR-10 dataset. QAT is implemented using uniform quantization of weights and activations, with both constant and dynamic scaling factors considered. Constant scaling applies a fixed factor across activations, whereas dynamic scaling recomputes scaling factors adaptively during inference.

Noisy training, in contrast, directly injects additive Gaussian noise into activations during training, with the same distribution applied during inference. This exposes the model to realistic perturbations and compels it to learn representations that remain robust under noisy computations.

To quantify robustness, we use the *midpoint noise level* μ, defined as the noise standard deviation at which accuracy falls to the midpoint between its clean baseline and random guessing (cf. Sec. 6.1). This metric provides a stable measure of tolerance against noise and allows direct comparison across models and training strategies.

The evaluation is performed by measuring accuracy under globally injected noise during inference, fitting the accuracy–noise curve, and extracting μ. This enables a systematic comparison of robustness for different bit-widths, scaling strategies, and training methods.

6.3.2 Findings

The comparison between QAT and noisy training reveals several characteristic patterns.

Quantization-aware training. Quantization alone provides a modest degree of robustness, but the effect strongly depends on bit-width and the scaling strategy. For LeNet-5 on CIFAR-10, 8-bit quantization achieves higher robustness than 4-bit (Fig. 6.6), since larger dynamic ranges reduce clipping effects. Constant

Table 6.2 Robustness of VGG-11 and ResNet-18 on CIFAR-10 under different bit widths and scaling factors using QAT. Constant scaling yields higher robustness than dynamic scaling, and increasing the scaling factor improves tolerance up to a point where the network collapses completely. Reproduced with permission from [7].

Model	Bitwidths	Scaling factors	Peak accuracy (%)	Midpoint noise level μ
VGG-11	fp32	-	**87.7**	0.154 (±0.5%)
	8-bit	dynamic	87.2	0.024 (±0.1%)
		0.5	84.3	0.145 (±0.2%)
		1	82.2	0.2 (±0.2%)
		2	76.8	0.222 (±0.3%)
		3	10.0	0.013 (±0.0%)
	4-bit	dynamic	86.5	0.031 (±0.1%)
		0.5	84.5	0.12 (±0.2%)
		1	82.6	0.177 (±0.2%)
		2	78.3	**0.23 (±0.3%)**
		3	10	0.010 (±0.1%)
ResNet-18	fp32	-	87.0	0.495 (±0.2%)
	8-bit	dynamic	**87.3**	0.452 (±0.3%)
		0.5	87.1	0.526 (±0.3%)
		1	87.0	0. 557 (±0.3%)
		4	86.5	0.577 (±0.3%)
		8	85.7	0.625 (±0.4%)
		10	10	0.05 (±2.5%)
	4-bit	dynamic	86.8	0.281 (±0.3%)
		0.5	86.5	0.492 (±0.4%)
		1	86.7	0.605 (±0.3%)
		4	86.5	0.657 (±0.5%)
		8	86.5	**0.665 (±0.3%)**
		10	10	0.005 (±1.3%)

scaling factors can significantly increase the midpoint noise level μ, but at the cost of lower clean accuracy, leading to a trade-off between robustness and peak performance. Dynamic scaling preserves clean accuracy, but generally results in lower robustness. The Pareto trade-off between peak accuracy and robustness is highlighted in Fig. 6.6(c), and extends to deeper architectures such as VGG-11 and ResNet-18 (Table 6.2). Overall, QAT improves tolerance to noise but does not fundamentally change the fragility of deeper models.

Noisy training. In contrast, noisy training proves highly effective across all architectures. By exposing the network to Gaussian perturbations during training, the learned representations become resilient to noise injected during inference. As shown in Fig. 6.6, noisy training consistently raises robustness far beyond that of quantization, while maintaining competitive clean accuracy. Table 6.3 summarizes this effect across LeNet-5, VGG-11, and ResNet-18: the midpoint noise level μ improves substantially when noisy training is applied, largely independent of model scale or parameter count.

Table 6.3 Overall comparison of architectures on CIFAR-10. ResNet-18 exhibits higher robustness than LeNet-5 and VGG-11, owing to its skip connections, which compensate for the increased depth that otherwise amplifies noise sensitivity. Across all models, noisy training effectively hardens networks against noise. Reproduced with permission from [7].

Property	**LeNet-5**	**VGG-11**	**ResNet-18**
FLOPS (M)	0.66	276.56	37.53
Param (M)	0.06	132.86	11.69
Noisy layers	12	27	33
Peak accuracy (%)	75	87.7	86.9
Robustness μ w/o Noisy Training	0.286	0.154	0.494
Robustness μ with Noisy Training	2.59	2.957	3.023

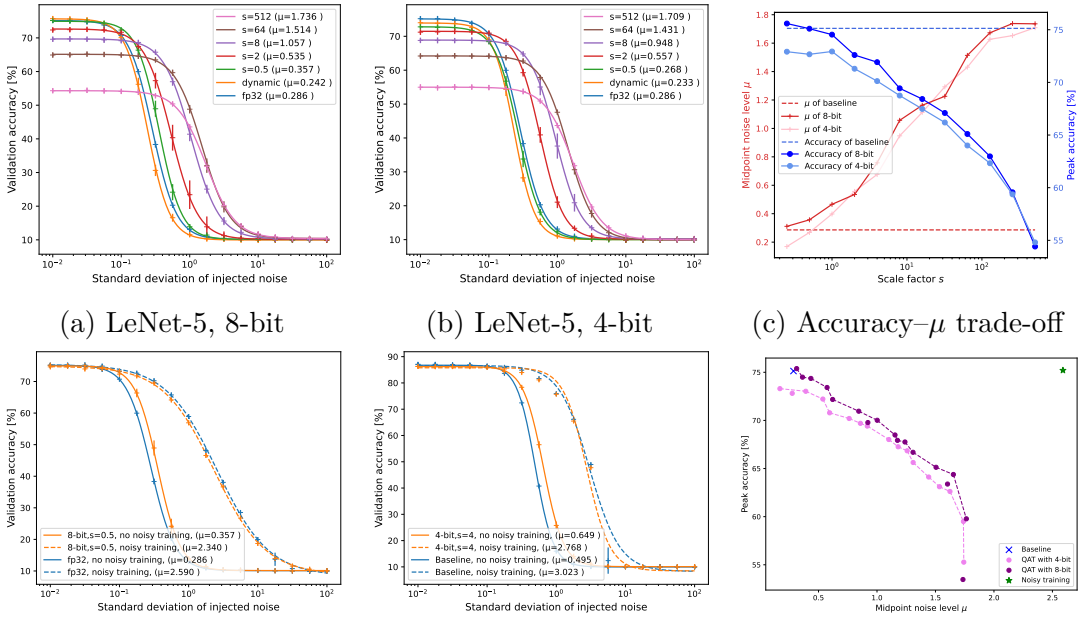
Combining methods. When quantization is combined with noisy training, robustness remains dominated by the effect of noisy training (Fig. 6.6). Quantization does not further increase robustness, but it does enable more compact models without losing the benefits of noise-aware training. This makes noisy training with quantization a practical approach for robust deployment on accelerators with limited precision.

Summary. In summary, QAT and noisy training both contribute to robustness, but in different ways. QAT shifts the trade-off between accuracy and tolerance by controlling scaling, whereas noisy training fundamentally reshapes network representations to withstand noisy inference. The combination yields compact and robust models, but noisy training emerges as the more decisive hardening method.

6.4 Variance-aware Noisy Training

The previous section showed that noisy training is the most effective strategy to harden neural networks against noisy computations. However, it also assumes that the noise distribution observed during training is identical to the one encountered at inference time. This assumption rarely holds for analog accelerators: noise levels drift with temperature, voltage, device aging, and even between individual chips. As a result, models trained with noisy training at a fixed noise level often fail when the actual operating conditions deviate.

To address this limitation, we introduce Variance-Aware Noisy Training



(a) LeNet-5, 8-bit (b) LeNet-5, 4-bit (c) Accuracy–μ trade-off

(d) LeNet-5 + noisy training (e) ResNet-18 + noisy train. (f) Pareto frontier

Figure 6.6 Robustness of LeNet-5 and ResNet-18 on CIFAR-10 under quantization and noisy training. Top row: quantization alone, showing the effect of bit width, scaling factors, and the trade-off between accuracy and midpoint noise level μ. Bottom row: quantization combined with noisy training, demonstrating the substantial robustness gains across architectures and the resulting Pareto frontier. The dramatic improvement in robustness of quantized models with noisy training is clearly visible. All x-axes are logarithmic. Reproduced with permission from [7].

(VANT). The key idea is to extend noisy training by injecting noise with varying strength, sampled from a distribution rather than fixed at a single level. In this way, models are explicitly exposed to the variability characteristic of real hardware, and can learn representations that remain robust even under fluctuating noise conditions.

6.4.1 Methodology

Previous results (Figs. 6.7a, 6.7b) show that noisy training substantially outperforms both QAT and perturbation-based methods such as Sharpness-Aware Minimization (SAM) [39], which improves generalization by guiding gradient descent towards flatter minima. They also reveal a critical limitation: robustness is only achieved when the noise level at inference closely matches the fixed noise level used during training. As soon as the two deviate, accuracy collapses rapidly (Fig. 6.7c). This motivates VANT, which explicitly accounts for variability in noise strength over time and across devices.

Standard noisy training injects Gaussian noise with a fixed standard deviation, thereby assuming that the accelerator's noise is constant across devices and stable over time. In practice, however, analog accelerators exhibit fluctuations due to temperature, voltage, and device drift, such that fixed-noise training does not match inference conditions.

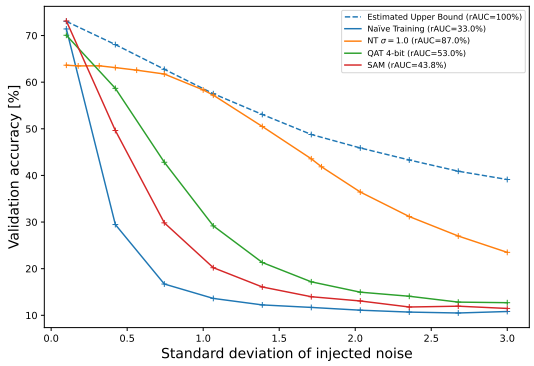
To address this, VANT samples the injected noise level itself from a distribution:

$$\begin{aligned} x &\sim \mathcal{N}(0, \sigma_{\text{var}}), \\ \sigma_{\text{var}} &\sim \mathcal{N}(\alpha \cdot \sigma_{\text{train}}, \theta), \end{aligned} \tag{6.3}$$

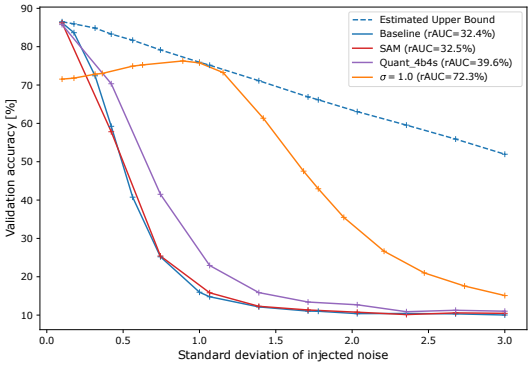
where σ_{train} is the nominal noise level of the target hardware, α calibrates the sampled distribution to σ_{train}, and θ controls its variability. During training, σ_{var} is sampled per input, and noise x is injected additively into activations in the forward pass, exposing the network to a range of perturbation levels that mimic real hardware conditions.

Robustness is quantified using the relative area under the accuracy–noise curve (rAUC). Let $A(\sigma)$ denote the accuracy under noise level σ. We compute

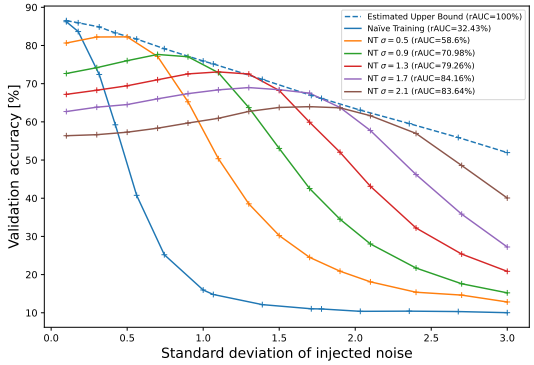
$$rAUC = \frac{\int A(\sigma)\, d\sigma}{\int A_{\text{ideal}}(\sigma)\, d\sigma}, \tag{6.4}$$



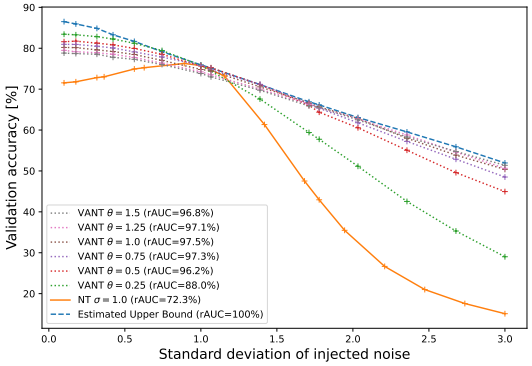
(a) LeNet-5



(b) ResNet



(c) Noisy training with different training



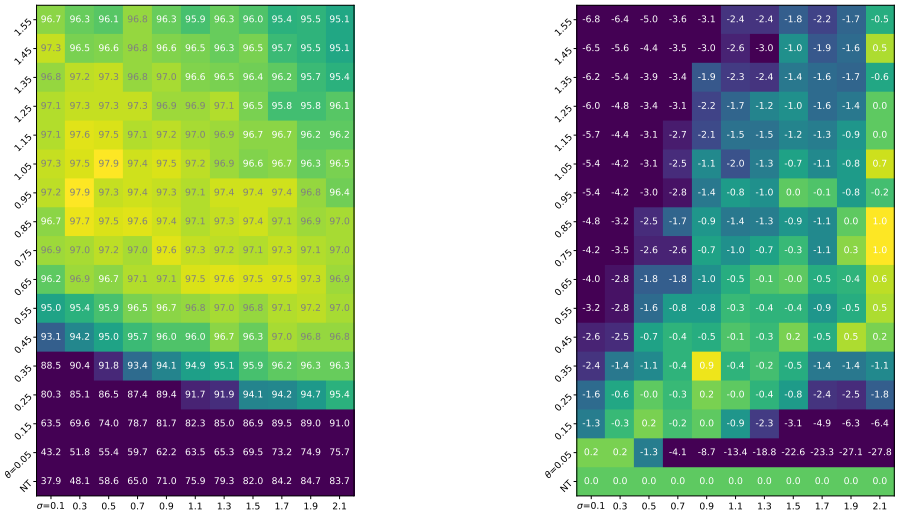
(d) VANT with varying θ vs. noisy training noise.

Figure 6.7 Comparison of hardening methods on CIFAR-10. (a) LeNet-5 and (b) ResNet show that noisy training substantially outperforms quantization and SAM. However, this robustness strongly depends on training and inference noise levels matching. (c) ResNet under varying inference noise: each noisy training curve peaks where training and inference noise coincide, while the dashed line marks the upper bound with perfectly matched noise. (d) VANT with different variability levels θ, demonstrating higher robustness across broader noise ranges compared to standard noisy training. Adapted from [8].

where the denominator represents the ideal case of perfectly matched training and inference noise. The rAUC is normalized between 0 and 1 and measures how closely a method approaches this upper bound of robustness.

6.4.2 Findings

The comparison of hardening methods in Figs. 6.7a and 6.7b confirms that noisy training substantially outperforms both QAT and SAM in terms of robustness. However, Fig. 6.7c illustrates a key limitation: each noisy training curve peaks precisely at the noise level used during training, and robustness collapses as



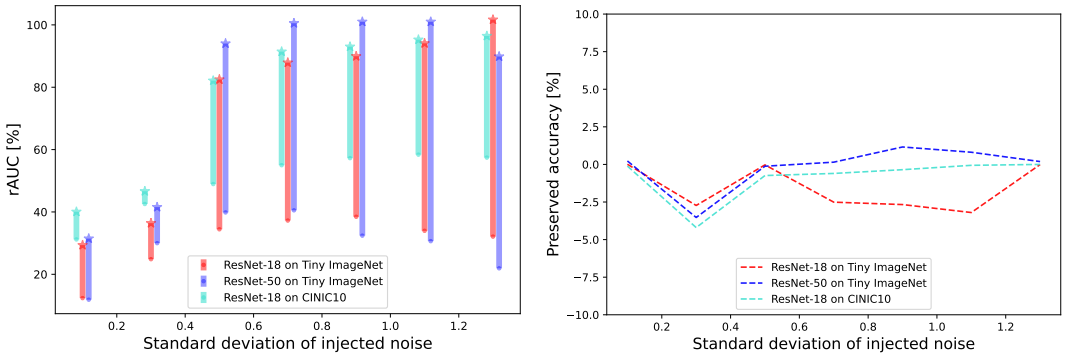
(a) Robustness (rAUC %) (b) Preserved Accuracy

Figure 6.8 Heatmaps illustrating how the variability parameter θ influences robustness and accuracy of VANT on CIFAR-10 with ResNet-18. (a) Robustness measured as rAUC increases with higher θ. (b) Preserved accuracy compared to standard noisy training decreases if variability becomes too large. Together, these plots show how robustness and accuracy trade off under different noise environments and provide guidance for selecting suitable hyperparameters. Reproduced with permission from [8].

soon as inference noise deviates. The dashed line shows the theoretical upper bound, corresponding to perfectly matched training and inference noise. This demonstrates that while noisy training is highly effective, it is also brittle under realistic conditions where noise is not static.

VANT directly addresses this limitation. By sampling the training noise level from a distribution, the network is exposed to a range of perturbation strengths and learns representations that remain stable under variation. Figure 6.8 illustrates how robustness depends on the choice of the variability parameter θ and the baseline hardware noise level σ_{train}. While larger values of θ generally increase robustness (rAUC), too much variability reduces preserved accuracy. An approximately linear relation $\theta \approx 0.4 \cdot \sigma_{\text{train}}$ was found to balance these effects, yielding robust yet accurate models. Figure 6.7d illustrates that while VANT remains sensitive to the variability parameter, it is significantly more robust across a broader range of noise levels than plain noisy training.

The benefits of VANT generalize across datasets and architectures. As shown in Fig. 6.9, it enhances robustness on more complex datasets such as CINIC-10



(a) Robustness (rAUC) (b) Preserved Accuracy

Figure 6.9 Generalization of VANT to more complex datasets and deeper architectures. (a) Robustness (rAUC) improves consistently across CINIC-10 and Tiny ImageNet for both ResNet-18 and ResNet-50. (b) Preserved accuracy remains close to standard noisy training, with only minor deviations. These results demonstrate that VANT reliably increases robustness across tasks and model scales, while incurring little accuracy cost. Reproduced with permission from [8].

and Tiny ImageNet, and scales effectively to deeper architectures like ResNet-50. Crucially, VANT sustains high robustness even under strong and varying noise levels, where standard noisy training rapidly deteriorates.

In conclusion, VANT overcomes the brittleness of standard noisy training by explicitly modeling noise variability. It provides stable robustness across datasets, architectures, and operating conditions, marking an important step toward sustainable deployment of analog accelerators. While already highly effective, further refinements may extend its applicability even more broadly.

Summary

This chapter developed a progressive view on robustness in neural networks for analog accelerators, moving from diagnosis to hardening, and finally to methods that address real-world variability.

First, *Walking Noise* showed that robustness is not a uniform property but emerges through heterogeneous, layer-specific mechanisms such as weight scaling under additive perturbations and self-binarization under multiplicative noise. This diagnostic perspective highlighted the opportunity of targeted countermeasures rather than uniform approaches.

Second, we compared systematic hardening strategies against noisy computations. While QAT can trade accuracy for tolerance, noisy training proved to be the most effective method for achieving robustness under static noise conditions, outperforming both QAT and perturbation-based baselines such as SAM.

Finally, we introduced VANT, a training strategy that extends noisy training by explicitly modeling variability in noise levels. Instead of assuming a fixed noise distribution, it samples noise strengths during training, exposing the network to the fluctuations characteristic of analog hardware. This variance-aware formulation makes robustness an inherent property of the learned representations, rather than a brittle consequence of matched training and inference conditions.

Taken together, these contributions trace a coherent trajectory: robustness can be understood at the layer level, strengthened with effective hardening, and sustained by accounting for the variability inherent to analog hardware. This progression lays the foundation for practical deployment of neural networks on analog accelerators, and points toward future refinements such as adapting methods to further device imperfections and extending beyond simple Gaussian noise to richer distributions and multiple injection points along the computational chain. At the same time, the results highlight that even simple noise models can be remarkably effective, suggesting that tailored yet computationally inexpensive hardening strategies remain a promising direction.

Part II

Accelerating Bayesian Neural Networks

7

Bayesian Neural Networks

Uncertainty is the only certainty there is, and knowing how to live with insecurity is the only security.

— John Allen Paulos

Uncertainty lies at the heart of real-world machine learning. Sensors produce noisy signals, environments evolve unpredictably, and training data can never fully capture the diversity of the world. When neural networks are deployed in safety-critical domains such as autonomous driving, medical diagnostics, or embedded control, it is therefore not sufficient to provide single deterministic outputs. Instead, models must also provide a measure of confidence in their outputs, enabling downstream systems to react appropriately: they may defer to a human operator, trigger a safe fallback mechanism, or request additional information when predictions are unreliable.

Conventional deep neural networks, despite their outstanding predictive accuracy, are poorly equipped to serve this role. Outputs are typically interpreted through the softmax function, which is often miscalibrated and tends to produce overconfident predictions even on inputs far outside the training distribution [148].

This structural inability to quantify uncertainty has become a central obstacle to the trustworthy deployment of deep learning [40]. A common failure mode is that networks issue highly confident but wrong predictions on out-of-distribution inputs. In such cases, models lack the ability to express *I do not know*, a capability that is crucial for robust decision-making.

A principled approach to modeling uncertainty is provided by Bayesian neural networks (BNNs). By replacing deterministic weights with probability distributions, BNNs extend classical neural networks into a probabilistic framework [234], [242]. Predictions thus become distributions rather than fixed values, capturing both data variability and model uncertainty. This perspective was consolidated by Jospin, Laga, Boussaid, Buntine, and Bennamoun [28], who provide a comprehensive overview of Bayesian inference strategies and a unified workflow for BNNs.

The theoretical foundation of BNNs is mathematically rigorous, but exact Bayesian inference over high-dimensional weight distributions is intractable. Approximate approaches such as sampling-based or variational inference methods require multiple posterior samples and forward passes, resulting in high runtime and memory demands [23], [234]. Consequently, while BNNs provide the most principled approach to uncertainty estimation, they are also among the most resource-intensive models to train and deploy.

Within the structure of this work, this chapter marks the transition from deterministic computation to probabilistic machine learning. It introduces the foundations of uncertainty quantification, surveys the main Bayesian inference strategies for BNNs, and discusses their computational challenges. The overarching message is clear: reliable uncertainty estimation is not an auxiliary feature but a fundamental capability, and BNNs provide the mathematical basis upon which this capability can be built. Later chapters build on these foundations: first by deploying the Probabilistic Forward Pass on embedded systems to demonstrate how closed-form propagation of an extreme SVI approximation can replace costly sampling in BNN inference, then by introducing ensemble-based methods as practical Bayesian approximations, and finally by exploring photonic accelerators where hardware noise itself becomes a controllable source of stochasticity.

7.1 Quantifying Uncertainty

Uncertainty quantification provides the foundation for probabilistic modeling. It captures not only how confident a model is in its predictions, but also why predictions may be uncertain. In practice, two principal forms of uncertainty are distinguished [23], [28], [153], [223].

Aleatoric uncertainty. Aleatoric uncertainty refers to the inherent randomness of the data-generating process. It arises from sources such as sensor noise, ambiguous labels, or intrinsic variability of physical systems. Since it is tied to the data itself, it cannot be reduced by collecting additional observations.

Epistemic uncertainty. Epistemic uncertainty corresponds to modeling error: it reflects incomplete knowledge about the data-generating process. This includes insufficient or unrepresentative training data, structural misspecification of the model class, or imperfections in the training procedure. Formally, it is represented by the posterior distribution $p(\theta \mid D)$, which quantifies the range of plausible parameter values given the data. Unlike aleatoric uncertainty, epistemic uncertainty can be reduced by collecting more representative data or by improving the model. It is particularly pronounced for inputs that lie outside the training distribution, making it crucial for out-of-distribution detection and active learning.

Metrics. In BNNs for regression tasks, aleatoric uncertainty is often modeled explicitly using an *aleatoric head*. Following Kendall and Gal [153], each regression target is represented by two outputs in the final layer: one predicts the mean $\mu(x)$ and the other the variance $\sigma^2(x)$, which models input-dependent Gaussian noise. This heteroscedastic Gaussian formulation jointly learns predictive means and variances, enabling the model to capture heterogeneous noise levels across the input domain. For numerical stability, it is common to predict $\log \sigma^2(x)$ instead of $\sigma^2(x)$ directly. Such aleatoric heads are specific to regression, since classification outputs are categorical and their uncertainty is naturally expressed through the distribution over class probabilities rather than explicit variance terms.

In classification, uncertainty is commonly quantified using information-theoretic metrics. Let x denote an input with label $y \in \{1, \dots, C\}$, and let the predictive distribution be defined by marginalizing over the posterior,

$$p(y \mid x, D) = \int p(y \mid x, \theta)\, p(\theta \mid D)\, d\theta. \tag{7.1}$$

The total predictive uncertainty is measured by the Shannon entropy of this distribution [253],

$$H[y \mid x, D] = -\sum_{c=1}^{C} p(y = c \mid x, D) \log p(y = c \mid x, D). \tag{7.2}$$

This quantity can be decomposed into contributions from aleatoric and epistemic sources [41], [116]. The expected data uncertainty, also known as softmax entropy (SME), isolates the aleatoric part,

$$\mathbb{E}_{p(\theta|D)}\Big[H[y \mid x, \theta]\Big] = -\frac{1}{N} \sum_{n=1}^{N} \sum_{c=1}^{C} p(y = c \mid x, \theta_n) \log p(y = c \mid x, \theta_n), \tag{7.3}$$

while the mutual information (MI) between predictions and parameters captures the epistemic part,

$$I[y, \theta \mid x, D] = H[y \mid x, D] - \mathbb{E}_{p(\theta|D)}\Big[H[y \mid x, \theta]\Big]. \tag{7.4}$$

It should be noted, however, that this decomposition is approximation-dependent and its exact interpretation remains debated in the community [2], [41].

Beyond accuracy and uncertainty decomposition, the *calibration* of predictive probabilities is a key indicator of reliability. A well-calibrated model outputs confidence values that correspond to the true likelihood of correctness, ensuring trustworthy uncertainty estimates. To quantify calibration, two metrics are widely used.

The negative log-likelihood (NLL) evaluates how well predicted probabilities align with observed labels,

$$\mathrm{NLL} = -\frac{1}{N} \sum_{n=1}^{N} \log p(y_n \mid x_n, D), \tag{7.5}$$

with lower values indicating better calibration. The expected calibration error

(ECE) measures the gap between predicted confidence and empirical accuracy by binning predictions [148],

$$\mathrm{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{N} \Big| \operatorname{acc}(B_m) - \operatorname{conf}(B_m) \Big|, \tag{7.6}$$

where B_m is the set of samples in bin m, $\operatorname{acc}(B_m)$ is their accuracy, and $\operatorname{conf}(B_m)$ their average confidence.

The area under the receiver operating characteristic curve (AUROC) is used to quantify separation between in-domain and out-of-distribution data. It integrates the true positive rate (TPR) over the false positive rate (FPR) across thresholds,

$$\mathrm{AUROC} = \int_0^1 \mathrm{TPR}(\mathrm{FPR}^{-1}(x))\, dx.$$

An AUROC of 1.0 indicates perfect separation, while 0.5 corresponds to random guessing [228].

Together, these metrics provide a principled way to assess both the decomposition of predictive uncertainty and the reliability of probability estimates. They also illustrate why probabilistic approaches are attractive: they offer a unified way to model uncertainty, distinguish its sources, and evaluate prediction quality. To realize these metrics in the context of neural networks, BNNs provide the natural mathematical framework, as they extend deterministic models into a probabilistic setting where both aleatoric and epistemic uncertainty can be quantified in a principled manner.

7.2 Bayesian Neural Networks: Foundations

Bayesian neural networks extend classical neural networks by placing probability distributions over their parameters. Instead of learning a single point estimate θ^* for the weights, a BNN treats weights as random variables and infers a posterior distribution given observed data D [234]. By Bayes' rule [23], the posterior is expressed as

$$p(\theta \mid D) = \frac{p(D \mid \theta)\, p(\theta)}{p(D)}, \tag{7.7}$$

where $p(\theta)$ is a prior distribution, $p(D \mid \theta)$ the likelihood, and $p(D)$ the marginal likelihood or evidence.[1] The prior reflects assumptions or domain knowledge about the weights, while the posterior captures updated beliefs after observing data [28], [216]. The evidence $p(D)$ serves as a normalizing constant but is intractable to compute for neural networks, since it requires integrating over all possible parameter configurations.

Predictions for a new input x are obtained by marginalizing over the posterior,

$$p(y \mid x, D) = \int p(y \mid x, \theta)\, p(\theta \mid D)\, d\theta, \tag{7.8}$$

which defines the posterior predictive distribution. This connects directly to the uncertainty measures introduced in the previous section: predictive entropy reflects total uncertainty, while its decomposition into aleatoric and epistemic parts depends on the likelihood and posterior, respectively.

The Bayesian perspective contrasts with the frequentist view. In frequentist inference, parameters θ are fixed but unknown, and learning corresponds to estimating them via maximum likelihood or related criteria. Bayesian inference instead treats parameters as random variables with a prior distribution, and learning corresponds to updating this belief in light of data. This perspective yields predictive distributions that naturally quantify both aleatoric and epistemic uncertainty [23], [28].

The posterior $p(\theta \mid D)$ involves high-dimensional integrals that lack closed form and are costly to approximate reliably, necessitating the use of approximate inference methods in practical BNNs. Sampling-based approaches, such as Markov chain Monte Carlo, provide asymptotically exact posterior draws but scale poorly with model size. Optimization-based approaches, such as variational inference, trade statistical fidelity for computational scalability, typically relying on tractable Gaussian approximations. More expressive variational families or advanced samplers can improve posterior fidelity but at substantially higher computational cost.

The overall process of BNN training and inference is illustrated in Figure 7.1, following the perspective of Jospin, Laga, Boussaid, Buntine, and Bennamoun [28]. It consists of three stages:

[1] The rule is named after Reverend Thomas Bayes, whose posthumous essay [255] first described the principle.

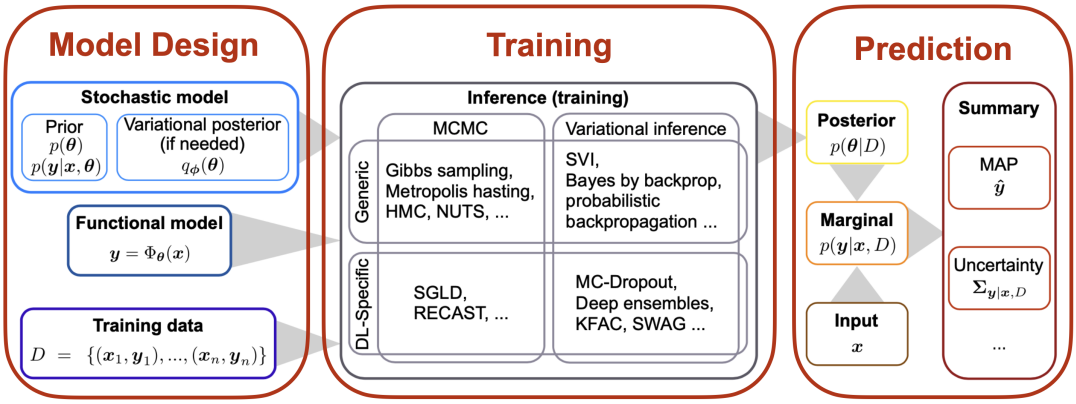


Figure 7.1 Workflow of a BNN. Design specifies both the architecture and prior distributions. Training applies Bayesian inference to approximate the posterior. Prediction marginalizes the posterior to form the predictive distribution, from which point estimates and uncertainty metrics can be derived. Adjusted from [28].

(i) **Design:** the neural architecture and prior distributions are specified.

(ii) **Training:** Bayesian inference techniques approximate the posterior distribution.

(iii) **Prediction:** the posterior is marginalized to form the posterior predictive distribution $p(y \mid x, D)$.

To obtain uncertainty estimates, multiple forward passes are performed by sampling from the posterior, yielding a distribution of predictions from which metrics such as variance, SME, MI, or predictive Shannon entropy can be derived.

BNNs thus provide a principled probabilistic extension of neural networks by treating parameters as random variables. They yield posterior predictive distributions that naturally capture both aleatoric and epistemic uncertainty. While conceptually elegant, their exact inference is computationally infeasible for modern architectures, motivating the approximate methods discussed in the next section.

7.3 Bayesian Inference

7.3.1 Markov chain Monte Carlo (MCMC)

Markov chain Monte Carlo methods provide the classical foundation of Bayesian inference and remain the gold standard for drawing asymptotically exact samples

from complex posteriors [217]. They construct a Markov chain whose stationary distribution is the posterior $p(\theta \mid D)$ and generate samples $\{\theta_n\}_{n=1}^{N}$ from this chain, from which posterior expectations can be approximated by averaging. This makes MCMC a conceptually simple yet broadly applicable tool, albeit one whose computational cost typically increases rapidly with dimensionality.

The most widely known instance is the Metropolis algorithm [250] and its generalization by Hastings [248]. The key idea of Metropolis–Hastings is to construct a proposal distribution $q(\theta' \mid \theta)$ for candidate moves and to accept a proposal with probability

$$\alpha(\theta, \theta') = \min\left(1,\, \frac{p(\theta' \mid D)\, q(\theta \mid \theta')}{p(\theta \mid D)\, q(\theta' \mid \theta)}\right). \tag{7.9}$$

This accept–reject step, together with the Hastings correction for asymmetric proposals, enforces detailed balance and guarantees that the posterior $p(\theta \mid D)$ is the stationary distribution of the chain. The defining property of Metropolis–Hastings is that it only requires ratios of posterior densities, which ensures that the intractable evidence $p(D)$ cancels out. An acceptance rate strictly below one prevents the chain from collapsing onto the maximum a posteriori mode and ensures that the full posterior distribution, rather than only its peak, is represented.

Running MCMC in practice requires careful handling of initialization and convergence. The initial part of each chain, known as the burn-in or warm-up phase, is discarded because early samples are still influenced by the arbitrary starting point rather than the stationary distribution [212]. During warm-up, proposal distributions or other tuning parameters are often adapted to improve efficiency, after which sampling proceeds with fixed settings to ensure validity. Convergence and sample quality are then assessed using standard diagnostics: autocorrelation functions and the effective sample size (ESS) quantify the amount of independent information contained in a chain [238], while the potential scale reduction statistic $\hat{R}$ compares within-chain to between-chain variance to detect lack of convergence [50], [237].

Despite its generality, classical Metropolis–Hastings suffers from poor mixing in high-dimensional parameter spaces. For models such as BNNs, naive proposals lead to low acceptance rates and highly correlated samples, which results in prohibitively slow convergence. To address this limitation, gradient-informed

methods exploit local geometry. Hamiltonian Monte Carlo (HMC), originally introduced as Hybrid Monte Carlo [245] and popularized in machine learning by Neal [220], is inspired by classical physics and augments the parameter space θ with auxiliary momentum variables r. It defines a Hamiltonian consisting of potential and kinetic energy,

$$\mathcal{H}(\theta, r) \;=\; U(\theta) + K(r) \;=\; -\log p(\theta \mid D) + \tfrac{1}{2} r^\top M^{-1} r\,, \tag{7.10}$$

where $U(\theta)$ is the potential energy given by the negative log posterior and $K(r)$ is the kinetic energy with mass matrix M. In this Bayesian interpretation, the posterior landscape acts as the potential energy surface, and by sampling random momenta r, the algorithm follows approximate Hamiltonian trajectories that traverse this landscape. Proposals are generated by simulating these dynamics with the leapfrog method using step size ε and trajectory length L. Since integration is approximate, a Metropolis accept–reject step at the end of each trajectory corrects for numerical error and ensures that the posterior remains the stationary distribution. This accept–reject step is essential for validity and is what distinguishes HMC from mere gradient-based optimization dynamics. The resulting proposals are long and directed, suppress random-walk behavior, and substantially improve mixing in continuous, high-dimensional spaces.

The No-U-Turn Sampler (NUTS) addresses the need to set the trajectory length L by dynamically expanding paths until a U-turn criterion is met and simultaneously adapting the step size ε during warm-up using dual averaging [206]. This removes the need for manual tuning and makes NUTS the default choice in many modern probabilistic programming frameworks.

Practical advice. In the context of BNNs, manual tuning of HMC parameters is particularly challenging. Selecting a suitable trajectory length L and step size ε is often infeasible in practice: inappropriate choices can either trap the chain in highly correlated states or cause excessive rejections, and the sensitivity grows with network size. NUTS has proven to be a real game changer in this setting. By adapting path lengths and step sizes automatically during warm-up, it makes HMC one of the easiest inference methods to apply reliably, despite its high computational cost. In practice, we found that NUTS consistently yields stable results for BNNs without the extensive parameter tuning otherwise required.

Limitations. Nevertheless, MCMC remains challenging for large, modern neural network architectures. Each proposal in MCMC requires at least one evaluation of the model likelihood or its gradient with respect to all network parameters. For HMC, this cost scales with the trajectory length L, since each trajectory requires L gradient evaluations. NUTS can be even more expensive, as it explores trajectories by recursively building a binary tree until a U-turn condition is met, leading to up to $2L-1$ gradient evaluations per iteration [206]. Multiple chains, warm-up, and the need for many effectively independent samples further amplify both compute and memory demands. As a result, MCMC is mainly applicable to small-scale models, where it serves both as a practical tool and as a rigorous reference for evaluating approximate inference methods. Its prohibitive computational and memory cost in modern architectures contrasts with the more scalable approximations discussed next.

7.3.2 Variational Inference (VI)

While MCMC provides asymptotically exact posterior samples, its computational demands scale poorly with model size. Variational inference offers a scalable alternative by recasting Bayesian inference as an optimization problem. The idea is to introduce a family of tractable distributions $q_\phi(\theta)$, parameterized by variational parameters ϕ, to approximate the intractable posterior $p(\theta \mid D)$ [144], [176], [231]. A seminal application to neural networks is *Bayes by Backprop* [189], which demonstrated variational learning of Gaussian weight distributions and laid the foundation for scalable approximate Bayesian deep learning. Instead of drawing samples directly from the posterior, the goal is to find the member of this family that is closest to the true posterior according to a divergence measure.

The most common choice is the Kullback–Leibler divergence (KL), which for two distributions $q(\theta)$ and $p(\theta)$ is defined as [176], [251]

$$\mathrm{KL}(q(\theta) \,\|\, p(\theta)) \;=\; \mathbb{E}_{q(\theta)}\left[\log \frac{q(\theta)}{p(\theta)}\right] \;=\; \int q(\theta) \log \frac{q(\theta)}{p(\theta)}\, d\theta. \tag{7.11}$$

Although asymmetric, the KL divergence is non-negative and equals zero if and only if $q(\theta) = p(\theta)$ almost everywhere.

Directly minimizing $\mathrm{KL}\big(q_\phi(\theta) \,\|\, p(\theta \mid D)\big)$ is infeasible, since the true posterior $p(\theta \mid D)$ contains the intractable evidence term $p(D)$. Instead, the problem is

reformulated in terms of the *evidence lower bound (ELBO)*, which is a tractable lower bound on the marginal log-likelihood $\log p(D)$. Maximizing the ELBO is equivalent to minimizing the KL divergence, but avoids the need to compute $p(D)$ explicitly [144], [176], [231]:

$$\mathcal{L}(\phi) \;=\; \mathbb{E}_{q_\phi(\theta)}\Big[\log p(D \mid \theta)\Big] - \mathrm{KL}\Big(q_\phi(\theta)\,\|\,p(\theta)\Big). \tag{7.12}$$

The first term encourages data fit, while the second term regularizes the approximation towards the prior.

Optimization of the ELBO can be carried out with stochastic gradient methods. In this setting, stochastic variational inference (SVI) employs mini-batches of data and stochastic gradient estimates to scale variational inference to large datasets [213]. The reparameterization trick [29], [209], first applied to Bayesian neural networks in *Bayes by Backprop* [189], enables low-variance gradient estimates by rewriting stochastic sampling as a deterministic transformation of parameters and auxiliary noise. For instance, if $q_\phi(\theta) = \mathcal{N}(\theta \mid \mu, \sigma^2)$, then sampling can be expressed as

$$\theta = \mu + \sigma\,\epsilon, \qquad \epsilon \sim \mathcal{N}(0,1), \tag{7.13}$$

which disentangles the randomness ϵ from the variational parameters (μ, σ) and allows gradients to propagate through them efficiently. Intuitively, this transformation lets gradients bypass the stochastic sampling step and flow into the variational parameters, making optimization with backpropagation feasible.

The expressiveness of the variational family $q_\phi(\theta)$ critically determines the quality of the approximation. Mean-field Gaussians, which assume independence across parameters, offer an attractive compromise: they are simple, scale efficiently to large models, and often yield sufficiently good uncertainty estimates in practice [144], [176]. Despite their tendency to underestimate posterior uncertainty, they remain widely used due to their efficiency and ease of implementation.

More expressive families can be employed when higher fidelity is required. Matrix-variate Gaussians, for example, introduce correlations across rows and columns of weight matrices, which is particularly natural for convolutional filters where nearby weights tend to co-vary. Normalizing flows instead transform simple base distributions through sequences of invertible mappings, allowing the capture of multimodal or skewed posteriors [23], [201]. These richer families

yield more faithful approximations, but at the cost of additional parameters and computational overhead, since each transformation must be optimized alongside the base network.

Practical advice. SVI-trained BNNs are highly sensitive to hyperparameter choices, and poor configurations often lead to degenerate solutions. If the posterior variance is initialized too high, the network quickly learns that predicting "I do not know" is a valid strategy and may never recover to produce meaningful outputs. Several strategies have proven effective in practice to mitigate this issue:

- **Warm-starting the means.** Initializing the posterior means from a pre-trained deterministic model provides a strong starting point for predictions.
- **Underestimating the prior variance.** Explicitly initializing prior variances significantly smaller than the means biases the model toward confident predictions in early epochs, preventing collapse into trivial uncertainty.
- **Balancing KL and data fit.** In practice, the KL term in the ELBO must be down-weighted relative to the likelihood. Too much weight causes posterior collapse into high uncertainty, while too little leads to overfitting and ignoring the prior. Finding this balance is among the most sensitive hyperparameters in SVI.
- **KL annealing.** Replacing the fixed balance with a dynamic schedule, where the KL weight is gradually increased during training, has proven particularly effective. Linear annealing is a common choice. While this approach reduces sensitivity to initialization, the final balance between KL and likelihood terms still requires hyperparameter tuning.

Together, these strategies stabilize optimization and substantially improve the reliability of SVI-trained BNNs, though at the cost of increased hyperparameter search.

Compared to sampling-based methods, VI trades asymptotic exactness for computational scalability. It enables deterministic optimization procedures with fast convergence and amortized inference. Its scalability on GPUs and compatibility with modern deep learning toolchains explain why SVI has become the standard variational approach for training large-scale BNNs.

Beyond Classical Bayesian Inference

The discussion so far has focused on the two principal methods for Bayesian inference: MCMC, with HMC and NUTS as its most effective variants for high-dimensional BNNs, and VI, with SVI as the key scalable instantiation. These approaches represent the classical Bayesian route to training BNNs: asymptotically exact sampling on the one hand, and optimization-based approximations on the other.

Beyond these methods, more pragmatic approximations have become popular in deep learning. Monte Carlo Dropout (MCDO) [175] and Deep Ensembles (DEs) [154] trade theoretical rigor for ease of use and empirical effectiveness. They will be revisited in Chapter 9, which broadens the perspective to ensemble-based methods. For a comprehensive overview of approximate Bayesian inference in BNNs, we refer to Jospin, Laga, Boussaid, Buntine, and Bennamoun [28].

7.4 Evaluation Datasets

Before evaluating the practical behavior of different inference methods, it is essential to clarify what constitutes a meaningful assessment of uncertainty estimation. In deterministic models, performance can often be summarized by a single accuracy or loss value, whereas for BNNs, evaluation must capture not only predictive quality but also how well epistemic and aleatoric uncertainty are represented. This places specific requirements on the datasets used for analysis. Ideally, they should provide clear distinctions between in- and out-of-distribution samples, expose ground-truth aleatoric variability, and remain low-dimensional enough to allow direct visualization of predicted uncertainties. At the same time, suitable quantitative metrics are needed to compare uncertainty quality across methods in a consistent and interpretable manner.

While low-dimensional datasets are invaluable for interpreting model behavior and visualizing uncertainty, they do not reflect the full complexity of realistic machine learning tasks. A comprehensive evaluation of BNNs therefore requires benchmarks that jointly address interpretability, scalability, and diversity in task type. To this end, we employ three complementary datasets that together span these requirements. The *Noisy Sine* benchmark provides a one-dimensional regression task with explicit ground-truth uncertainty, enabling direct comparison

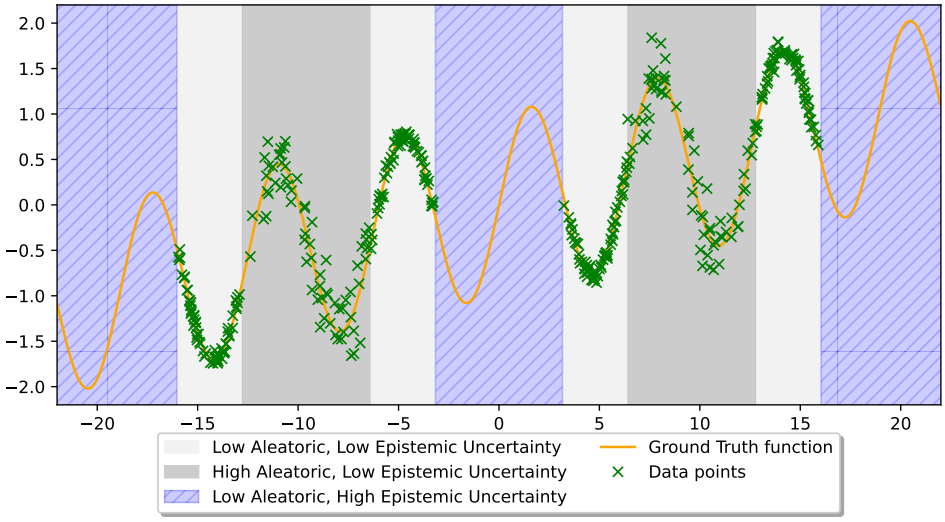


Figure 7.2 Schematic of the Noisy Sine dataset showing the generative function $f(x)$, data-bearing regimes with different Gaussian noise levels (aleatoric), and data-free gaps that induce epistemic uncertainty. Evaluation points are placed at regime midpoints and gap midpoints. Reproduced with permission from [6].

between predicted and true aleatoric variance. The *Two Half Moons* dataset introduces a low-dimensional classification problem that remains simple enough for visualization but exposes the challenge of epistemic uncertainty both near decision boundaries and with increasing distance from the training data. Finally, the *Dirty-MNIST* benchmark extends the analysis to high-dimensional image classification, allowing us to assess whether findings from the simpler settings generalize to more realistic and complex data domains.

7.4.1 Noisy Sine

The *Noisy Sine* benchmark is a one-dimensional regression task designed to probe both aleatoric and epistemic uncertainty in a controlled setting. The generative function is

$$y = \sin(x) + 0.05x + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma^2(x)), \tag{7.14}$$

where ϵ represents aleatoric uncertainty in the form of heteroscedastic Gaussian noise. The input domain is partitioned into multiple regimes, as illustrated in Figure 7.2. Some regimes contain training points and use distinct Gaussian noise levels, inducing different levels of *aleatoric* uncertainty. Other regimes contain no training points at all, which elicit high *epistemic* uncertainty. To establish

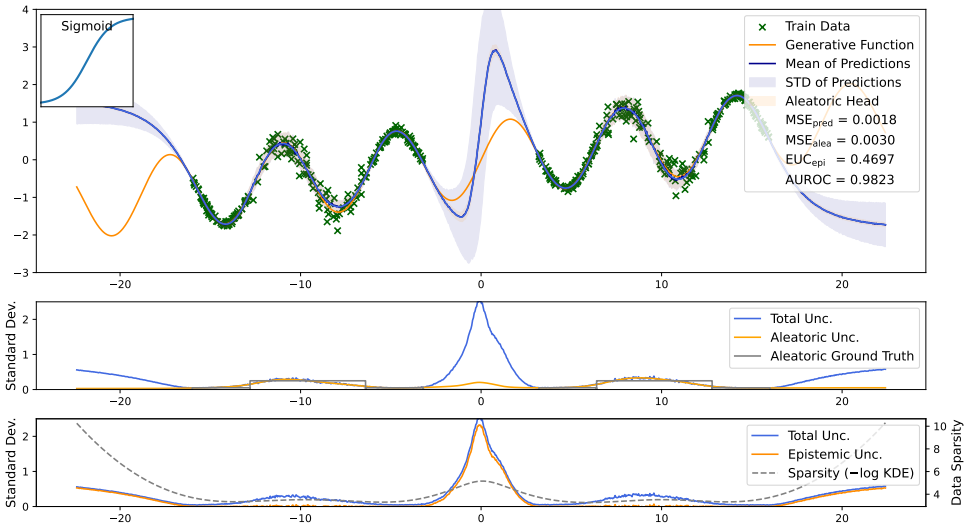


Figure 7.3 Visualization convention for the Noisy Sine benchmark. Each panel is divided into three rows: *Top:* predictive mean with a band for the *total* predictive standard deviation, overlaid with training samples. *Middle: aleatoric* uncertainty—predicted by the aleatoric head and compared against the ground-truth noise profile. *Bottom: epistemic* uncertainty together with the negative log KDE of the training data (dashed line), illustrating its correlation with data sparsity.

ground-truth uncertainty patterns, three regime types alternate along the input axis: (*i*) dense regions with low Gaussian noise $\sigma(x)$=0.05, (*ii*) mid-density regions with half as many samples and higher noise $\sigma(x)$=0.25, and (*iii*) data-free gaps that express purely epistemic uncertainty. This structure creates alternating segments where uncertainty is dominated by aleatoric effects within the data regimes and by epistemic effects in the gaps and beyond the training range.

Figure 7.3 illustrates the visualization layout used throughout this work. Each plot consists of three rows: (*top*) predictive mean with total uncertainty, (*middle*) predicted aleatoric variance alongside the ground-truth noise profile, and (*bottom*) epistemic uncertainty together with the reference sparsity of training data.

Prediction quality. Since the ground-truth is known, predictive accuracy can be measured by the mean squared error (MSE) between the predicted mean $\mu(x)$ and the noiseless ground-truth target $f(x) = \sin(x) + 0.05x$. Evaluation is restricted to data-bearing regimes, as extrapolation outside these regions is not meaningful for assessing fit.

Aleatoric evaluation. Since the ground-truth noise standard deviation $\sigma(x)$ is defined within each data regime, aleatoric uncertainty can be evaluated directly. We compute the MSE between the predicted STD $\hat{\sigma}(x)$ (from the aleatoric head) and the true injected STD $\sigma(x)$. Lower values indicate more accurate estimation of the heteroscedastic noise profile.

Epistemic evaluation. Assessing epistemic uncertainty is more difficult because no explicit ground truth exists. We therefore adopt the *Epistemic Uncertainty Coefficient (EUC)* proposed by Simonides [6]. The EUC is defined as the Pearson correlation between predicted epistemic uncertainty and the sparsity of training data, estimated by a Kernel Density Estimation (KDE). Given training samples $\{x_i\}_{i=1}^{N}$, the KDE at a point x with bandwidth h is

$$f(x) = \frac{1}{Nh} \sum_{i=1}^{N} K\Big(\frac{x - x_i}{h}\Big), \tag{7.15}$$

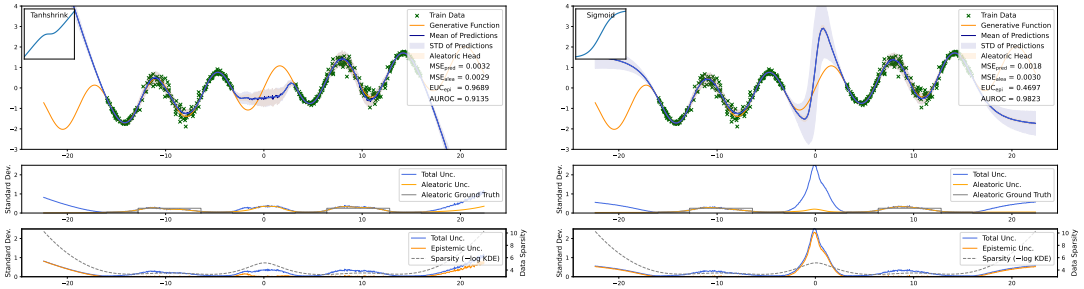
where K is typically chosen as a Gaussian kernel,

$$K(x) = \frac{1}{\sqrt{2\pi}} \exp\Big(-\tfrac{1}{2}x^2\Big). \tag{7.16}$$

Since well-covered regions correspond to high density and sparse regions to low density, epistemic uncertainty is expected to correlate with the *negative log density*. The EUC therefore measures whether the "shape" of the predicted epistemic uncertainty aligns with data coverage.

While intuitive, EUC has an important limitation: it does not capture absolute calibration. It is possible for a model to systematically underestimate uncertainty and still achieve a high EUC, as long as the spatial correlation with data density is preserved. For instance, we observed cases such as a BNN with a Tanhshrink activation where EUC reached the highest value among all experiments, yet visual inspection revealed clear underestimation of uncertainty (Figure 7.4).

Practical alternative. From a practitioner's perspective, the more relevant question is whether uncertainty estimates can reliably flag predictions outside the training domain. We therefore complement EUC with an area under the receiver operating characteristic curve (AUROC)-based evaluation of out-of-distribution detection. By labeling samples inside data regimes as in-distribution (ID) and



(a) Tanhshrink (EUC = 0.9689) (b) Sigmoid (EUC = 0.4697)

Figure 7.4 Illustration of the limitations of the Epistemic Uncertainty Coefficient. Panel (a) shows a model with Tanhshrink activation function, which achieved the highest EUC across experiments, yet clearly underestimates uncertainty in data-free regions. Panel (b) shows a model with Sigmoid activation function, which yielded a lower EUC but produced more realistic epistemic uncertainty. This highlights that EUC captures correlation with data sparsity but not absolute calibration, motivating the use of AUROC as a complementary metric.

those in gaps as out-of-distribution (OOD), we compute the AUROC to quantify how well epistemic uncertainty predictions separates the two cases. Unlike EUC, this metric evaluates the discriminative power of uncertainty for decision-making: a high AUROC indicates that the model provides a trustworthy signal of when its predictions should not be relied upon.

7.4.2 Two Half Moons

Following the one-dimensional regression task, the *Two Half Moons* dataset introduces a low-dimensional classification problem that remains simple enough for visualization while capturing key aspects of epistemic uncertainty. It is a two-dimensional binary classification benchmark with a non-linear decision boundary, frequently used to analyze how different methods capture uncertainty in classification settings. While aleatoric effects appear near the decision boundary, the different inference methods behave similarly in this region. Our evaluation therefore focuses on epistemic effects and explicit OOD detection relative to ID regions, where the methods diverge more clearly.

ID/OOD labeling. In contrast to the Noisy Sine benchmark, the *Two Half Moons* dataset does not provide ground-truth ID/OOD regions. We therefore construct reference labels via Kernel Density Estimation, similar to its use in the calculation of EUC, but here applied in the two-dimensional input space. Given

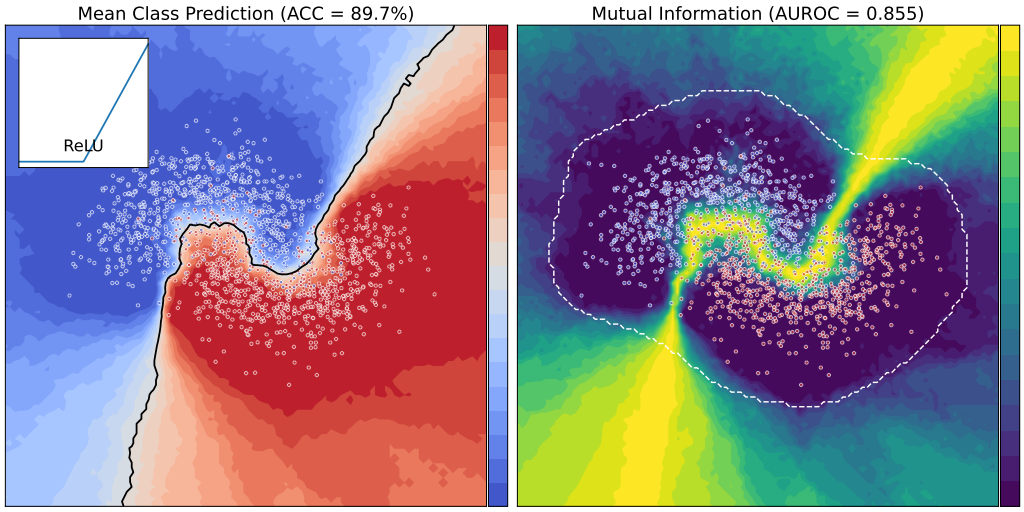


Figure 7.5 Visualization of the *Two Half Moons* dataset. Training samples are shown as small circles: blue for one class and red for the other. *Left:* mean class predictions with the non-linear decision boundary overlaid. *Right:* epistemic uncertainty quantified by mutual information, where higher values (see colorbar) indicate higher uncertainty. The dashed contour marks the OOD region obtained via KDE-based density thresholding. This setup enables evaluation of OOD detection quality via AUROC.

training samples $\{x_i\}_{i=1}^{N} \subset \mathbb{R}^2$, the density at a test point x is estimated as

$$\hat{f}(x) \;=\; \frac{1}{Nh^2} \sum_{i=1}^{N} K\Big(\frac{x - x_i}{h}\Big), \tag{7.17}$$

with Gaussian kernel $K(u) = \frac{1}{2\pi} \exp\Big(-\frac{1}{2}\|u\|^2\Big)$ and bandwidth h. The bandwidth is determined automatically using Scott's rule [243], a widely used heuristic that adapts to dataset size and dimensionality and provides stable density estimates without manual tuning. Binary ID/OOD labels are then obtained by thresholding this density at a chosen quantile: points above the threshold are treated as ID, while points below are labeled as OOD.

We quantify OOD detection by ranking test points with predicted epistemic uncertainty via mutual information and comparing against these ID/OOD labels using the AUROC. Figure 7.5 illustrates this setup: the left panel shows mean predictions and decision boundary, while the right panel shows epistemic uncertainty with the KDE-derived OOD contour.

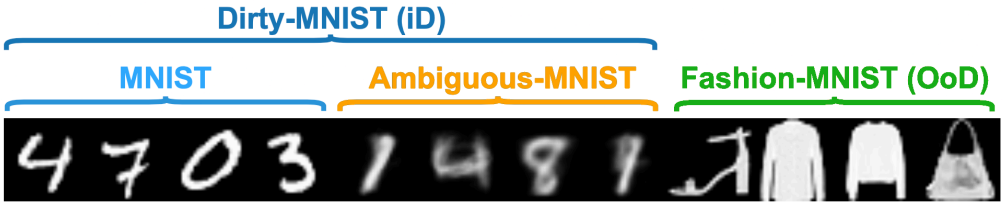


Figure 7.6 Schematic illustration of the Dirty-MNIST benchmark. From left to right: clean MNIST digits (ID), Ambiguous-MNIST samples generated by variational autoencoder interpolation (aleatoric uncertainty), and Fashion-MNIST images used as explicit OOD inputs (epistemic uncertainty). Figure adapted from [31].

7.4.3 Dirty-MNIST

The *Dirty-MNIST* benchmark extends the evaluation to high-dimensional image classification, providing a more realistic testbed for uncertainty estimation. It builds upon the classical MNIST dataset [232] and is specifically designed to probe both aleatoric and epistemic uncertainty in a controlled yet complex setting [31]. It combines three complementary subsets, which we collectively refer to as *Dirty-MNIST* in the following:

- **MNIST:** clean in-distribution digit images, serving as the reference domain where predictive uncertainty should remain low.
- **Ambiguous-MNIST:** digits generated by interpolating between latent codes of two MNIST digits using a variational autoencoder [208]. These samples are inherently ambiguous, making them a source of strong *aleatoric* uncertainty.
- **Fashion-MNIST:** an auxiliary dataset of clothing images [169], unseen during training and used as explicit OOD data. Predictions on these samples should exhibit high *epistemic* uncertainty.

Together, these subsets form a compact yet comprehensive benchmark. Following the setup of Mukhoti, Kirsch, Amersfoort, Torr, and Gal [31], models are trained jointly on MNIST and Ambiguous-MNIST, treating both clean and ambiguous digits as in-distribution data. MNIST provides the clean baseline, Ambiguous-MNIST emphasizes aleatoric effects, and Fashion-MNIST serves as the out-of-distribution probe for epistemic uncertainty. Figure 7.6 illustrates the three components side by side.

7.5 Empirical Insights into BNN Inference

The practical behavior of BNNs cannot be judged by theory alone. Even methods with solid theoretical foundations may succeed or fail depending on architectural choices and dataset characteristics. This sensitivity implies that a deterministic architecture cannot simply be made "Bayesian" by placing distributions on its weights; instead, small architectural adjustments must be considered to obtain reliable uncertainty estimates.

In the following, we analyze this effect on the *Noisy Sine* and *Two Half Moons* benchmarks. Together, these results highlight systematic patterns in how activation functions influence uncertainty quality, independent of raw predictive performance.

7.5.1 Impact of Activation Functions

One of the most striking observations across our experiments is that BNNs are highly sensitive to the choice of activation function. While predictive accuracy varies little across activation functions, uncertainty estimates differ substantially.

Experimental setup. All activation function experiments were conducted with an MLP of two hidden layers and 50 neurons each. We use HMC with NUTS, an asymptotically exact sampler whose adaptive warm-up reduces manual tuning and which serves as the de facto reference baseline for BNN inference quality. On the Noisy Sine dataset, we drew 1000 posterior samples after 1000 warm-up iterations, while for the Two Half Moons dataset we used 2000 samples with 2000 warm-up iterations to account for its higher complexity.

Noisy Sine. On the one-dimensional Noisy Sine task, BNNs with different activation functions generally fit the data regimes well and capture the injected heteroscedastic noise through the aleatoric head. Overall, however, the data fit is strong, as reflected in low MSE for most functions, with only a few clear outliers in both predictive means and aleatoric variance (Table 7.1).

The key differences arise in the epistemic component. Some activation functions, such as Sigmoid, produce smooth and moderate uncertainty growth with distance from the training data, whereas others, like Hardswish, exhibit erratic predictions and markedly inflated uncertainty bands.

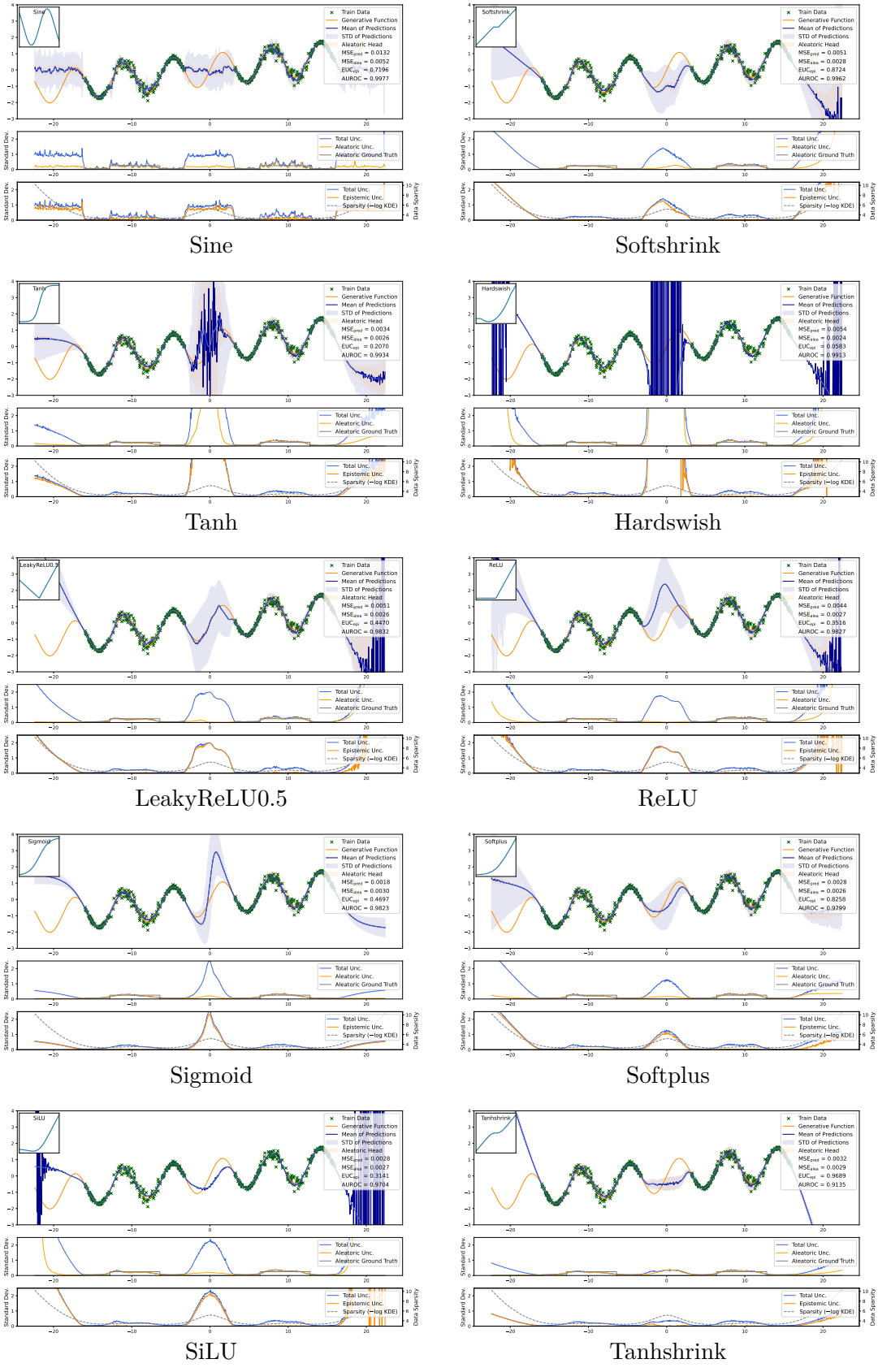


Figure 7.7 Comparison of activation functions on the Noisy Sine task, using MCMC-based BNNs. Uncertainty behavior differs markedly in OOD regions despite similar predictive fits, ranging from oscillatory and unstable to smooth but overconfident. Panels are ordered by decreasing AUROC.

Table 7.1 Activation functions on the Noisy Sine dataset, using MCMC-based BNNs. Metrics cover predictive accuracy, aleatoric variance estimation, and OOD detection quality.

Activation	MSE (Pred.) ↓	MSE (Aleatoric) ↓	EUC ↑	AUROC ↑
Cosine	0.0137	0.0057	0.6988	**0.9985**
Sine	0.0132	0.0052	0.7196	0.9977
Softshrink	0.0051	0.0028	0.8724	0.9962
Hardtanh	0.0066	0.0025	0.0978	0.9948
PReLU	0.0049	0.0028	0.4673	0.9935
Tanh	0.0034	0.0026	0.2070	0.9934
Hardswish	0.0054	0.0024	0.0583	0.9913
Hardsigmoid	0.0037	0.0035	0.7305	0.9906
Softsign	0.0042	0.0029	0.4661	0.9900
LeakyReLU	0.0050	0.0026	0.4801	0.9894
Mish	0.0050	0.0029	0.4195	0.9886
SELU	0.0053	**0.0024**	0.6794	0.9868
GELU	0.0029	0.0027	0.0443	0.9854
LogSigmoid	0.0024	0.0031	0.9264	0.9840
LeakyReLU0.5	0.0051	0.0026	0.4470	0.9832
ReLU	0.0044	0.0027	0.3516	0.9827
Sigmoid	**0.0018**	0.0030	0.4697	0.9823
Softplus	0.0028	0.0026	0.8258	0.9799
ELU	0.0040	0.0028	0.3459	0.9761
CELU	0.0040	0.0028	0.3459	0.9761
SiLU	0.0028	0.0027	0.3141	0.9704
RReLU	0.0351	0.0205	0.9284	0.9497
LogSoftmax	1.2008	0.5332	0.4433	0.9212
Tanhshrink	0.0032	0.0029	**0.9689**	0.9135
Softmin	0.2612	0.1641	0.1442	0.7065
Softmax	0.3499	0.2152	0.1170	0.6440
Hardshrink	0.2690	0.3039	0.0339	0.4760

Periodic activation functions (Sine, Cosine) form a special case: they yield consistently high uncertainties already a short distance away from the training data and therefore achieve the best AUROC values for OOD detection. This behavior can be attributed to their oscillatory structure, which tends to amplify epistemic uncertainty in regions beyond the training data. This benefit, however, comes at a slight cost to predictive MSE, reflecting the influence of their high-frequency components. The visualizations in Figure 7.7 also highlight the limitations of scalar metrics: for example, the Tanhshrink activation attains the highest EUC, yet clearly underestimates uncertainty in data-free regions. Nearly all activation functions ultimately succeed in distinguishing ID from OOD regions, but they differ markedly in how they express epistemic uncertainty once outside the data regimes.

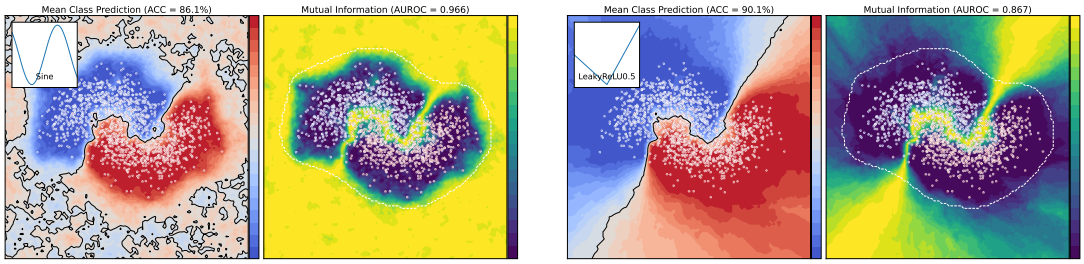
Two Half Moons. In the two-dimensional classification task, BNNs with different activation functions consistently capture the nonlinear decision boundary. However, they systematically misattribute the resulting ambiguity to epistemic rather than aleatoric uncertainty, which in theory should dominate in overlapping class regions. The crucial differences emerge in the outer OOD regions. Here, performance diverges sharply: periodic activation functions such as Sine and Cosine achieve excellent AUROC scores, displaying a sharp rise in uncertainty immediately beyond the data boundary, while some standard activation functions like Sigmoid remain overconfident even far outside the training support (Table 7.2, Figure 7.8). Across activation functions, we observe a characteristic ordering of OOD behavior:

(i) **Periodic** activation functions (Sine, Cosine) yield the strongest OOD discrimination, characterized by a sharp rise in uncertainty immediately outside the data regions.

(ii) **Smooth, positively sloped** activation functions on both sides of the origin (e.g., LeakyReLU0.5, SiLU, Hardswish) follow, typically increasing uncertainty with distance at a moderate rate.

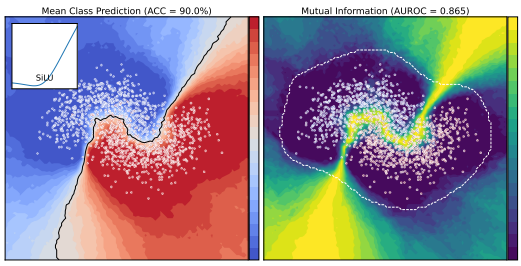
(iii) **One-sided or flatter** activation functions (e.g., ReLU, Softplus) tend to delay uncertainty growth, remaining confident deeper into OOD regions.

(iv) **Functions with shrinking response near the origin** (e.g., Softshrink, Tanhshrink) extend this behavior further.

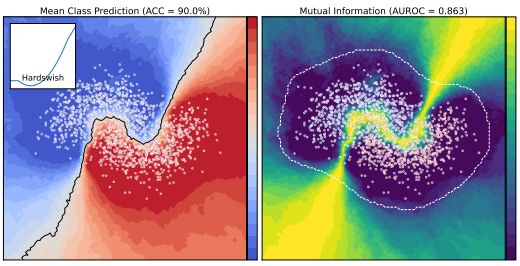


Sine

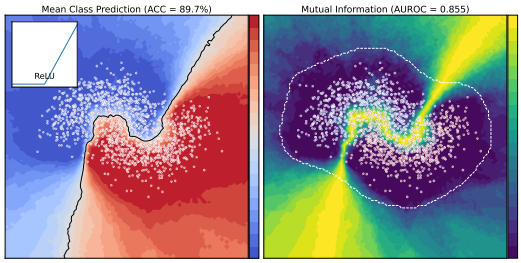
LeakyReLU0.5



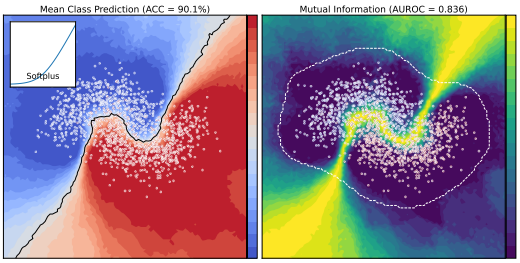
SiLU



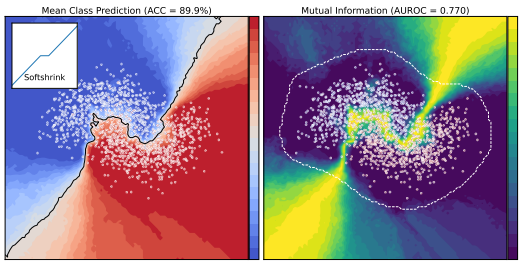
Hardswish



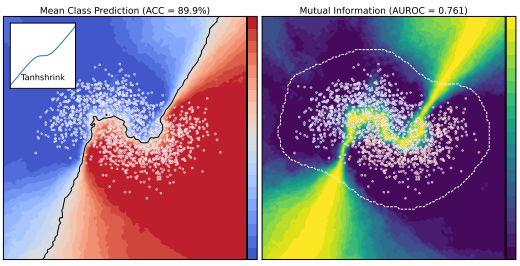
ReLU



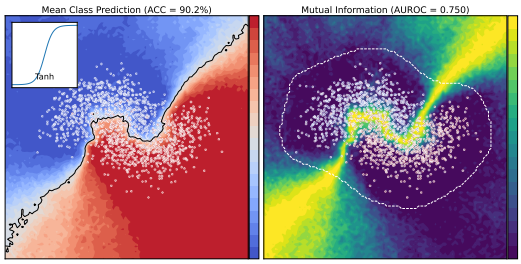
Softplus



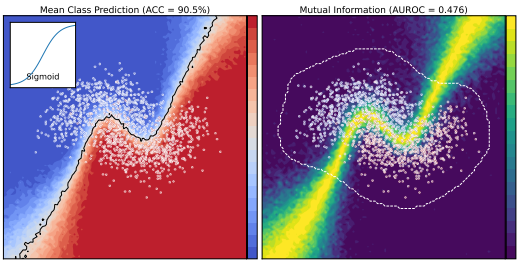
Softshrink



Tanhshrink



Tanh



Sigmoid

Figure 7.8 Representative uncertainty visualizations for different activation functions on the Two Half Moons dataset, using HMC-NUTS-based BNNs. Each panel shows mean predictions (left) with decision boundary and mutual information (right) as a proxy for epistemic uncertainty. While all predictions are uncertain at the decision boundary, the uncertainty increase with distance to the training data varies. Plots are ordered by decreasing AUROC.

(v) **Saturating sigmoidal** functions (e.g., Sigmoid, Tanh) are the most persistently overconfident.

Importantly, this ordering is *dataset-specific*: while it holds for Two Half Moons, it does not replicate on the Noisy Sine benchmark. This underlines that the relevance of a particular activation trait must be assessed in the context of the task rather than assumed to generalize universally.

A recent study by Tempczyk, Smoczyński, Smolenski-Jensen, and Cygan [33] highlighted the strong sensitivity of BNNs to activation choice in the context of mean-field variational inference. They argued that ReLU activation functions induce highly non-Gaussian posterior landscapes, which are poorly captured by Gaussian variational families, and showed that replacing ReLU with an optimized LeakyReLU substantially improves uncertainty calibration without compromising accuracy. Although our experiments rely on HMC with NUTS rather than variational inference, and are not limited to mean-field Gaussian distributions, the broader message carries over: activation functions exert a pronounced influence on uncertainty quality. To connect to this line of work, we evaluated their proposed best-performing variant—LeakyReLU with slope 0.5 in the negative region, denoted *LeakyReLU0.5* in our experiments. On the Noisy Sine task, its performance was nearly indistinguishable from standard ReLU and outperformed by the default LeakyReLU, indicating little advantage in this setting. In contrast, on the Two Half Moons dataset, LeakyReLU0.5 clearly outperformed both baselines and ranked among the strongest activation functions overall, surpassed only by the periodic functions. These results reinforce the view that carefully designed activation functions can mitigate uncertainty failures, while also confirming that their benefits remain highly task-dependent.

Insights and Takeaway. Across both benchmarks, several clear lessons emerge. First, predictive quality alone is not a reliable guide for architecture design: although most activation functions perform similarly within the data regime, their ability to capture uncertainty differs substantially. Second, it seems there is no universally best activation function. Functions that work well in one setting may fail in another, as exemplified by Sigmoid, which produces smooth and realistic uncertainty on Noisy Sine but remains overconfident in Two Moons. Third, certain patterns emerge: for example periodic activation functions often

excel at OOD detection in low-dimensional tasks. However, these tendencies are not universal and do not necessarily transfer between datasets.

The overarching message is that activation function choice is a critical design aspect in BNNs. Unlike in deterministic models, where activation functions can often be swapped without major impact, uncertainty-aware models are highly sensitive to this decision. Small-scale architecture searches are therefore essential to identify suitable functions for a given dataset and model. Periodic activation functions appear particularly promising for robust OOD detection, but no universal best option exists, and careful tuning remains indispensable.

Looking ahead, more adaptive strategies may further improve activation design for BNNs. In preliminary work, we explored Kolmogorov–Arnold Networks (KANs) [5] to learn task-specific activation functions directly from data. These methods showed encouraging potential to discover novel activation functions that outperform the standard set in terms of uncertainty quality for BNNs. Although this line of research is not yet mature enough to be included in the present work, it suggests that the future of BNN design may involve *learning* activation functions tailored to uncertainty quantification rather than relying on fixed, hand-crafted functions.

Other Architectural Effects. Beyond activation functions, other architectural choices also have a pronounced impact on uncertainty estimation in BNNs. Even within the same inference method, network depth and width strongly influence how uncertainty propagates. For MLP-based models, shallow or narrow networks tend to be overconfident, whereas overly large ones often become unstable and degrade in uncertainty quality. This reflects a practical trade-off: increasing capacity improves expressiveness but complicates training. Effective BNN design therefore requires balancing model capacity against the fidelity of uncertainty estimation.

A similar pattern appears across architectural families. Figure 7.9 compares a fully connected MLP (one hidden layer, 100 neurons) and a convolutional LeNet-5 on the Dirty-MNIST benchmark, both trained with mean-field SVI for 2000 epochs using KL annealing. Although predictive accuracy on MNIST is comparable (96.6% for the MLP, 97.9% for LeNet-5), the LeNet-5 achieves much stronger OOD detection (AUROC 0.93 vs. 0.75) and clearer uncertainty separation across the MNIST, Ambiguous-MNIST, and Fashion-MNIST subsets.

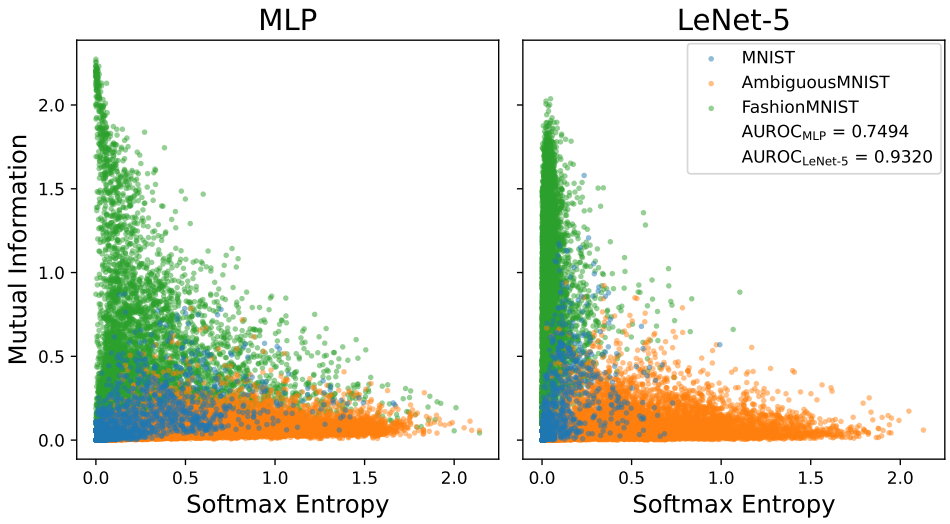


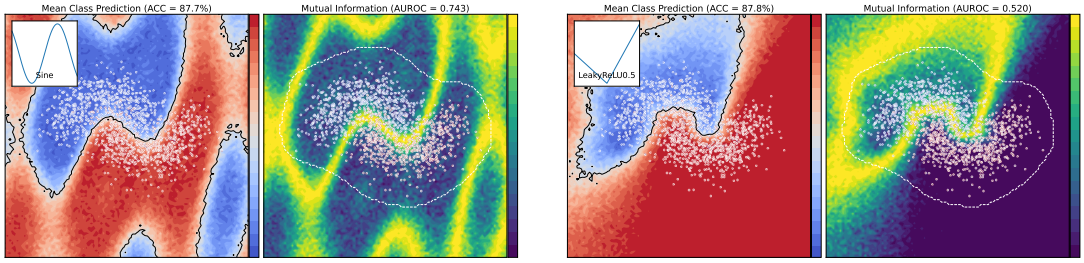
Figure 7.9 Comparison of two SVI-trained BNNs on the Dirty-MNIST benchmark: a one-hidden-layer MLP (100 units) and a LeNet-5 convolutional network. Uncertainty separation is illustrated in the space of softmax entropy (aleatoric uncertainty) and mutual information (epistemic uncertainty). The LeNet-5 achieves stronger OOD separation and better uncertainty calibration across MNIST (ID), Ambiguous-MNIST (aleatoric), and Fashion-MNIST (OOD) subsets, highlighting the impact of architectural expressiveness.

This highlights that greater architectural expressiveness—here through convolutional feature extraction—can substantially improve uncertainty estimation under identical inference settings. Hence, model architecture is a decisive factor in the calibration quality of BNNs.

7.5.2 Comparison of MCMC and SVI

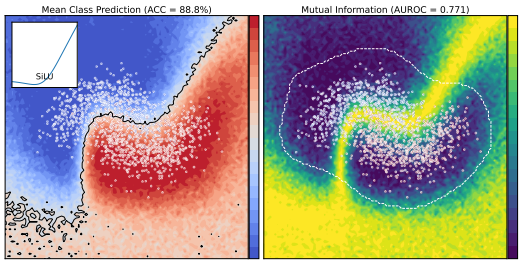
To compare the practical behavior of different Bayesian inference methods, we evaluate sampling-based inference using HMC with NUTS—referred to as MCMC in the following—against SVI on the *Two Half Moons* dataset. This setup enables a direct comparison between the posterior sampling approach and its variational alternative.

Comparison of Figure 7.10 with Figure 7.8 illustrates qualitative differences in the resulting uncertainty maps, while Table 7.2 summarizes their quantitative performance. As expected, MCMC generally achieves higher AUROC values for out-of-distribution detection and produces more stable uncertainty patterns across activation functions. Only in two cases does SVI reach superior AUROC, and in

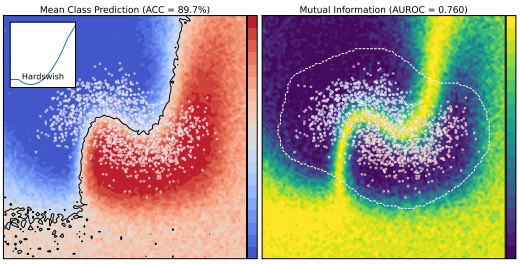


Sine

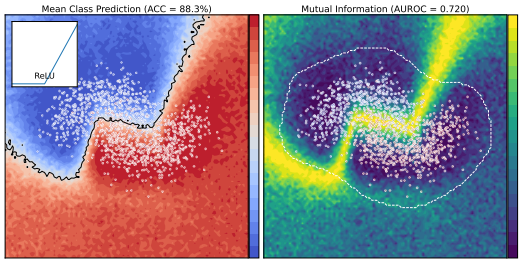
LeakyReLU0.5



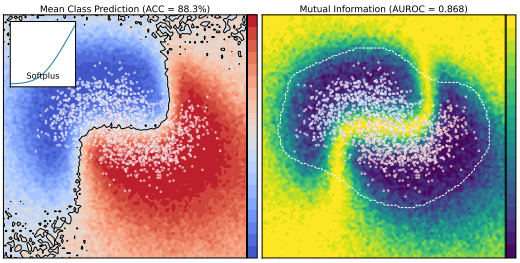
SiLU



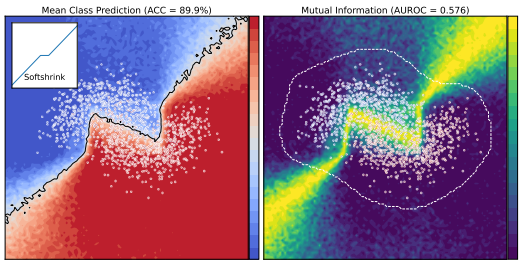
Hardswish



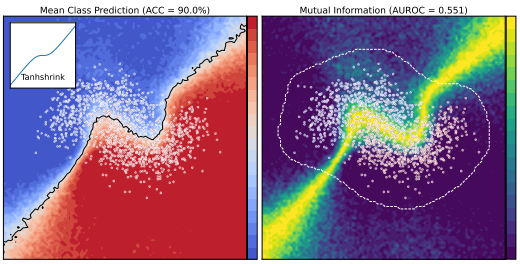
ReLU



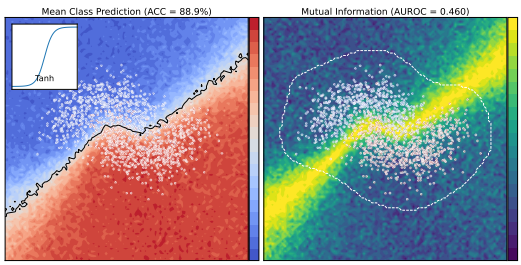
Softplus



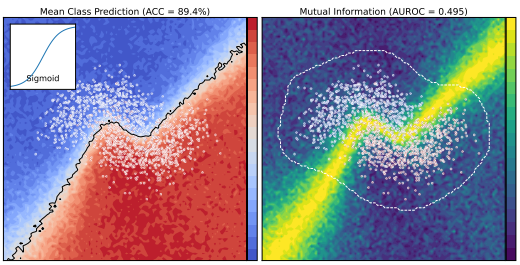
Softshrink



Tanhshrink



Tanh



Sigmoid

Figure 7.10 SVI-BNNs results on the Two Half Moons dataset with various activation functions. Trained with linear KL–annealing for 500 epochs after deterministic pretraining.

Table 7.2 Impact of activation functions on BNNs trained with SVI and MCMC on the Two Half Moons dataset. Accuracy (%) and AUROC are reported, with the better value per activation highlighted in bold.

Activation	**Accuracy [%] ↑**		**AUROC ↑**	
	SVI	**MCMC**	**SVI**	**MCMC**
Sine	**87.73**	86.13	0.7427	**0.9663**
LeakyReLU0.5	87.80	**90.07**	0.5198	**0.8666**
SiLU	88.80	**90.00**	0.7706	**0.8646**
Hardswish	89.67	**90.00**	0.7602	**0.8631**
ReLU	88.27	**89.67**	0.7205	**0.8551**
Softplus	88.27	**90.07**	**0.8680**	0.8361
Softshrink	**89.93**	89.87	0.5763	**0.7698**
Tanhshrink	**90.00**	89.93	0.5514	**0.7606**
Tanh	88.93	**90.20**	0.4605	**0.7498**
Sigmoid	89.40	**90.47**	**0.4953**	0.4757

merely three cases it slightly exceeds the predictive accuracy of MCMC. This confirms the expected advantage of sampling-based inference in both robustness and calibration.

Beyond average performance, the visual comparison reveals qualitative distinctions that highlight the greater sensitivity of SVI models to architectural choices—particularly the activation function. While the hyperparameter sensitivity of SVI is well known in the community, its dependence on activation functions has received far less attention. As discussed earlier in Section 7.5.1 and shown by Tempczyk, Smoczyński, Smolenski-Jensen, and Cygan [33], activation choice can substantially affect uncertainty calibration in BNNs. Their study, however, was limited to ReLU variants within mean-field SVI BNNs, whereas our experiments demonstrate that this sensitivity generalizes across a broader spectrum of activation functions and inference methods. The proposed *LeakyReLU0.5* [33] variant exhibited qualitatively distinct behavior but did not outperform the standard ReLU, underscoring that such effects are strongly task-dependent.

More broadly, our findings confirm that this phenomenon extends well beyond the ReLU family. Across activation functions, SVI outcomes vary considerably more than under MCMC, often leading to inconsistent or unstable uncertainty patterns. Similar activation shapes tend to induce similar artifacts: for instance, *LeakyReLU0.5* produces a pronounced asymmetry, leaving one region of the input space with persistently low uncertainty, while periodic activation functions such as *Sine* retain large-scale oscillatory patterns but lack the sharp ID/OOD

transition observed in the MCMC baseline. Interestingly, *Softplus* emerges as the best-performing activation under SVI, achieving AUROC values on par with the strongest MCMC models. This highlights a central observation: the optimal activation function for a given task and architecture is not invariant to the inference method. In practice, the most effective activation functions for SVI may differ from those for MCMC, complicating BNN research and the transfer of architectures between inference paradigms.

From a practitioner's perspective, these results have two main implications. First, SVI can, in principle, match the uncertainty quality of MCMC—but only under carefully tuned configurations. In this study, the Softplus–SVI combination reached comparable OOD detection quality, whereas most other activation functions underperformed the sampling-based models. Second, this tuning sensitivity emphasizes that SVI-based BNNs require substantially more effort in hyperparameter optimization and architecture adaptation to achieve reliable results. The stochastic optimization process and mean-field assumptions of SVI appear to amplify such sensitivities, in contrast to the more stable but computationally demanding MCMC approach.

For the higher-dimensional Dirty-MNIST benchmark [31], this trade-off becomes particularly evident. Figure 7.11 compares predictions from MCMC- and SVI-trained BNNs using a compact MLP with one hidden layer of 100 neurons. While both models achieve comparable accuracy, they struggle to disentangle epistemic from aleatoric uncertainty. Both assign high softmax entropy to Fashion-MNIST samples, but only the MCMC model simultaneously reports high mutual information, correctly identifying these samples as OOD. The SVI model, in contrast, misattributes many of them as aleatoric, resulting in weaker OOD discrimination.

A complementary experiment, shown in Figure 7.9, demonstrates that increased architectural expressiveness can mitigate these effects. Replacing the simple MLP with a convolutional LeNet-5—trained via SVI—leads to markedly improved uncertainty separation and OOD detection performance. Despite relying on approximate inference, the LeNet-5 achieves near-perfect separation between MNIST, Ambiguous-MNIST, and Fashion-MNIST subsets while maintaining high predictive accuracy. However, this architecture already lies in a regime where training with MCMC becomes prohibitively time-consuming.

Together, these results highlight a clear trade-off: MCMC remains the most

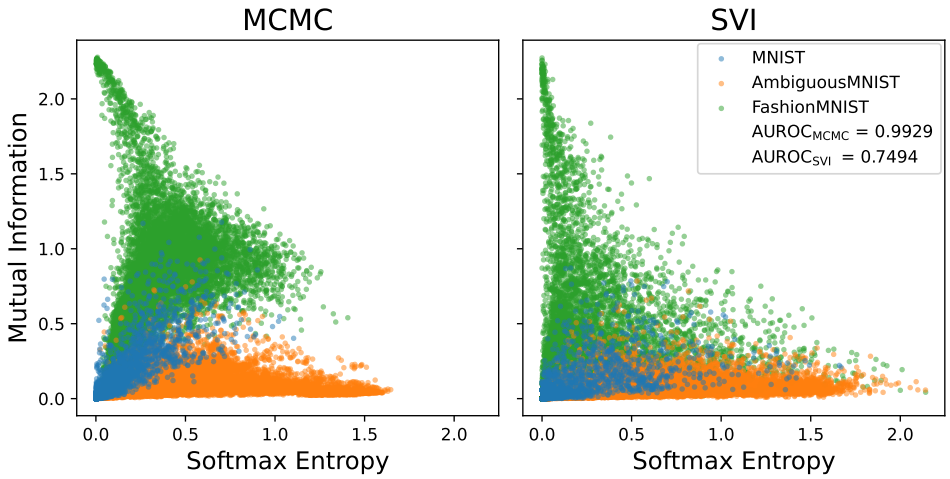
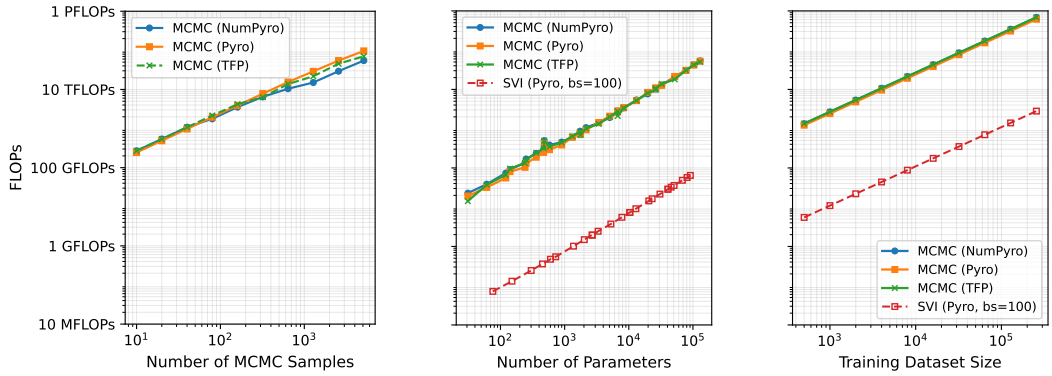


Figure 7.11 Comparison of MCMC- and SVI-trained BNNs on the Dirty-MNIST dataset using a MLP (one hidden layer, 100 neurons). Both models assign high softmax entropy to Fashion-MNIST samples—an incorrect attribution, since these should be characterized by high mutual information. The MCMC model however, also attributes high mutual information, correctly marking them as OOD, whereas SVI does in many cases not. Despite similar accuracy (95.8% MCMC vs. 96.6% SVI), MCMC delivers far superior OOD detection (AUROC MCMC = 0.99 vs. 0.75 for SVI).

reliable approach for uncertainty calibration but is computationally infeasible for more complex architectures, whereas SVI offers superior scalability and can leverage expressive models to compensate for its approximate nature.

Taken together, the three benchmarks reveal that the practical success of mean-field SVI for BNNs is highly dataset and architecture dependent. On simple, low-dimensional tasks such as *Noisy Sine* and *Two Half Moons*, the method often struggles to reproduce the well-calibrated uncertainty structure obtained with sampling-based inference, showing strong sensitivity to both architectural and hyperparameter choices. In contrast, on higher-dimensional datasets like *Dirty-MNIST*, where full MCMC becomes computationally infeasible, carefully tuned SVI models can achieve robust uncertainty separation and scale efficiently to larger networks. Overall, these results indicate that mean-field SVI remains a practical and scalable approximation, but its reliability—and the degree of tuning required—depend critically on the dataset characteristics and model architecture.



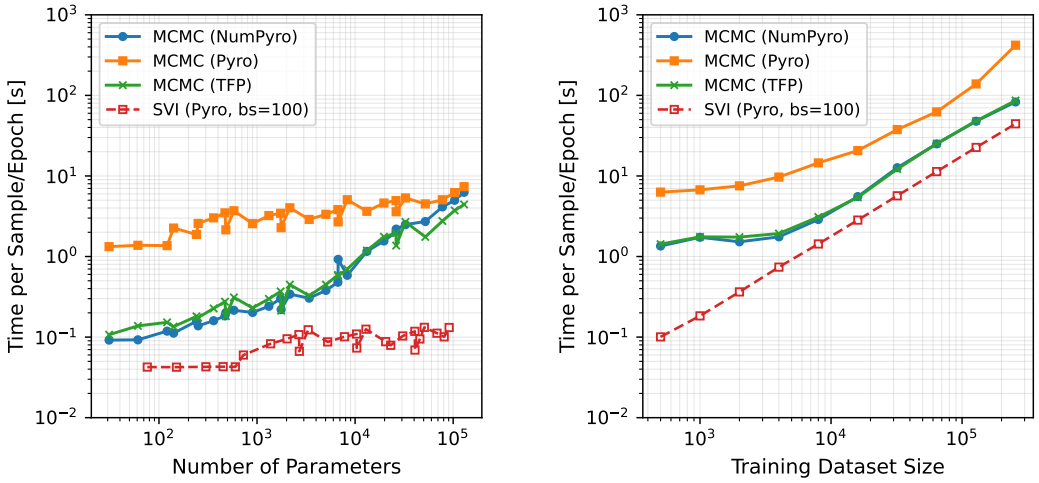
(a) MCMC Samples (b) Model Size (c) Dataset Size

Figure 7.12 Scaling of FLOPs for MCMC and SVI training of BNNs. FLOP counts increase linearly with both model and dataset size across all probabilistic programming frameworks (Pyro, NumPyro, TensorFlow Probability). While the overall trends are similar, SVI requires roughly two orders of magnitude fewer FLOPs, highlighting its superior computational efficiency.

7.5.3 Computational Scaling of Bayesian Inference

While previous sections have qualitatively noted that MCMC tends to be computationally more demanding than variational approaches, we now quantify this difference empirically. Specifically, we evaluate the computational scaling of HMC–NUTS and mean-field SVI across varying model and dataset sizes, as well as across three probabilistic programming frameworks: *Pyro* [82], *NumPyro* [98], and *TensorFlow Probability* [146]. The goal of these experiments is not to optimize predictive performance, but to characterize how computational cost grows with problem complexity. Here, we specifically measure the computational cost of *Bayesian inference* for *training* BNNs—that is, approximating the posterior distribution over network weights—rather than the subsequent *predictive inference* phase used for evaluating new inputs. This distinction, illustrated in Figure 7.1, is essential for understanding the trade-off between training and inference costs for BNNs: Approaches such as HMC are dominated by expensive posterior sampling during training, whereas other techniques may instead shift the computational burden toward a larger number of required samples—and thus forward passes—at inference time.

The scalability experiments in this subsection were conducted as part of a



(a) Model Size (b) Dataset Size

Figure 7.13 Average latency per sample (for MCMC) and per epoch (for SVI) across model and dataset size. While both methods scale with model size, SVI remains significantly faster, whereas MCMC incurs higher latency due to adaptive sampling steps.

master's thesis project.[2] All experiments were executed on an AMD EPYC 7302P 16-core processor, with hardware-level metrics collected using the `likwid` performance monitoring framework [221]. Unless otherwise stated, the evaluated networks were MLPs with three hidden layers of 50 neurons each. Model size was systematically varied by adjusting both the network depth (from one to five hidden layers) and layer width (from 25 to 200 neurons), while dataset size was scaled on the *Noisy Sine* benchmark by increasing the number of training samples. For MCMC, we employed HMC with the NUTS sampler and collected 50 posterior samples following 100 warm-up iterations. For SVI, models were trained for 150 epochs to ensure comparable computational budgets across settings. While these configurations are insufficient to achieve fully converged predictive quality, they provide a consistent and representative basis for measuring computational cost and scaling trends. All Pyro experiments were executed with JIT compilation enabled to mitigate Python interpreter overhead.

Across frameworks, the FLOP count increases approximately linearly with both model and dataset size (Figure 7.12). The differences between frameworks are negligible, indicating that they perform essentially the same amount of

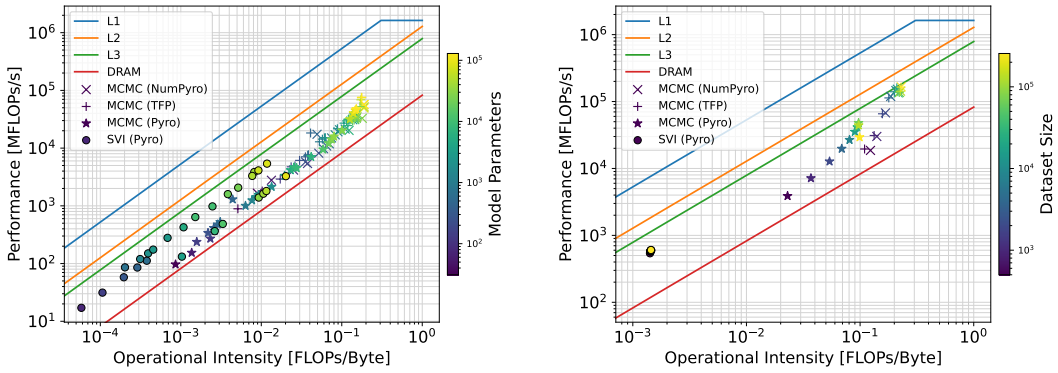
[2]At Heidelberg University and carried out by Jonathan Bernhard.

arithmetic work for comparable models. Overall, SVI requires about two orders of magnitude fewer operations than MCMC, consistent with the additional gradient-based sampling steps in HMC.

Figure 7.13 illustrates the measured latency per iteration—defined as time per MCMC sample or per SVI training epoch—as a function of model and dataset size. With increasing model size, the computational gap between the two inference schemes widens markedly: while SVI exhibits a gradual, near-linear growth in runtime, MCMC slows down disproportionately as the number of parameters increases. This trend is most evident for the NumPyro and TensorFlow Probability implementations, where time per sample rises steeply with model complexity. For Pyro, the effect is somewhat masked by interpreter overhead, which dominates total latency at smaller scales. Overall, the results highlight that MCMC scales considerably worse with parameter count than SVI, reinforcing the computational advantage of optimization-based variational inference.

We attribute the poorer scaling of MCMC—more precisely, HMC with the NUTS sampler—relative to SVI mainly to its reliance on repeated gradient evaluations for each posterior sample. Each trajectory in HMC involves simulating the system dynamics over many leapfrog steps, effectively multiplying the number of full forward and backward passes through the network. As model dimensionality increases, the posterior landscape becomes increasingly complex, demanding longer trajectories and smaller integration step sizes to maintain numerical stability. In contrast, SVI performs amortized inference by directly optimizing a variational objective, requiring only a single gradient update per mini-batch and scaling with only a modest increase in cost as model size grows. Empirically, this effect appears as a significantly steeper growth in computation time for larger architectures under MCMC.

A more detailed view of computational efficiency is provided by the roofline analysis in Figure 7.14. Across all configurations, MCMC sampling operates in a regime of low operational intensity, confirming that Bayesian sampling on modern CPUs is predominantly memory-bound rather than compute-bound. Most measurements fall between the effective DRAM and L3-cache bandwidths, indicating that limited data reuse and frequent memory transfers dominate runtime. Increasing either model or dataset size raises the operational intensity, as larger workloads improve cache locality and arithmetic-to-memory ratios. In



(a) Model Size (b) Dataset Size

Figure 7.14 Roofline analysis of MCMC and SVI training for BNNs. All configurations operate in the memory-bound regime between DRAM and L3-cache bandwidth, with larger models and datasets increasing operational intensity and improving hardware utilization.

particular, scaling the dataset shifts the computation closer to the L3-bandwidth limit, suggesting more efficient processor utilization for larger data regimes. From an efficiency standpoint, both larger models and datasets make better use of the available hardware, though at the cost of substantially higher absolute compute demand.

For SVI, the picture differs due to mini-batch training: all dataset-size configurations cluster around a similar operational intensity, as memory access patterns are dominated by fixed-size batches rather than the total dataset. When scaling model size, operational intensity increases proportionally with the number of parameters, yet remains below that of MCMC, reflecting the lower arithmetic density of its gradient-based optimization. Among the evaluated frameworks, NumPyro and TensorFlow Probability achieve the highest effective hardware utilization, while Pyro remains most constrained by interpreter and runtime overhead.

In summary, the scaling analysis highlights a clear computational trade-off. SVI consistently achieves lower runtime and scales more efficiently with both model and dataset size, whereas MCMC with NUTS incurs substantially higher cost but remains more robust in terms of convergence and calibration. All implementations operate in a memory-bound regime, indicating that future efficiency gains will primarily depend on improving data movement and cache utilization rather than on increasing raw compute throughput.

Summary

This chapter provided an empirical examination of Bayesian neural networks, analyzing how inference methods, architectural choices, and implementation factors shape uncertainty quality and computational efficiency. Across experiments, activation functions emerged as a major determinant of epistemic behavior: while predictive accuracy remained similar, uncertainty calibration varied drastically. Comparisons of MCMC and SVI showed that sampling-based inference yields the most reliable uncertainty estimates, whereas mean-field SVI is more sensitive to architecture and activation choice yet scales to larger models. Architectural expressiveness also plays a critical role, exemplified by the transition from a simple MLP to a slightly larger CNN, markedly improved uncertainty separation. Finally, computational scaling analyses revealed that SVI is orders of magnitude faster and scales better with model size than MCMC, though both are predominantly memory-bound.

Overall, reliable BNN inference depends as much on the network architecture and activation functions as on the chosen Bayesian inference method. MCMC remains the reference standard for small, precision-critical models, while SVI enables scalable Bayesian deep learning when coupled with well-chosen architectures.

While this chapter established the principles of designing and training Bayesian neural networks, deploying them on embedded hardware raises new challenges, where limited computation and memory dominate design trade-offs. The next chapters therefore move from training to deployment, exploring *efficient BNN inference* methods that sustain reliable uncertainty estimation under the tight constraints of real-world systems.

8

Compiling Probabilistic Forward Pass BNNs for Embedded Systems

In theory, there is no difference between theory and practice, while, in practice, there is.

— Benjamin Brewster

As discussed in the previous chapter, BNNs provide a principled way to quantify uncertainty, distinguishing between aleatoric and epistemic sources. However, nearly all established Bayesian inference methods, such as Markov chain Monte Carlo, stochastic variational inference, and even modern approximation techniques like Monte Carlo Dropout [175] or Deep Ensembles [154], remain computationally costly. In practice, reliable uncertainty estimation typically requires drawing many posterior samples and executing multiple forward passes per input, which prevents deployment on embedded and resource-constrained platforms. This chapter is based on our work *"Accelerated Execution of Bayesian Neural Networks using a Single Probabilistic Forward Pass and Code Generation"*, currently under review at the *ACM Transactions on Architecture and Code*

Optimization [3].

This chapter focuses on the Probabilistic Forward Pass (PFP) [48], [186], which can be regarded as an extreme form of SVI. While SVI models weights as Gaussian-distributed parameters under a mean-field assumption, PFP extends this approximation to activations as well. This assumption enables distributions to be propagated in closed form, allowing both predictions and uncertainties to be computed within a single forward pass. By eliminating repeated stochastic evaluations, PFP directly addresses the central computational bottleneck of BNNs. Although the Gaussian restriction reduces the ability to capture non-Gaussian activation distributions, it provides a tractable and efficient approximation particularly suited for inference on resource-constrained systems.

So far, PFP has remained largely theoretical, as its Gaussian-propagating operators are absent from standard ML frameworks. Without this support, no practical path to deployment was available. In this chapter, we close this gap by presenting the first end-to-end realization of PFP. We demonstrate how the deep learning compiler TVM [115], can be extended with custom operators to support PFP. These compilers provide a generic implementation pathway transferable across hardware backends while maintaining seamless integration with common ML workflows. As a result, models trained with SVI can be exported, optimized, and deployed with minimal effort—enabling a capability that was previously out of reach.

The chapter begins by establishing a training pipeline based on SVI and shows that PFP achieves comparable uncertainty estimation and out-of-domain detection to sampling-based baselines on the Dirty-MNIST dataset [31]. We then extend TVM with a library of custom operators for MLPs and CNNs, and apply both manual and automatic optimizations to improve the efficiency of computationally demanding operators. Finally, we benchmark the resulting implementation on embedded ARM processors, demonstrating speedups of up to four orders of magnitude compared to SVI-based baselines.

Altogether, this chapter demonstrates how deep learning compilers, operator optimizations, and algorithmic approximations together make BNN inference—and thus uncertainty quantification—feasible even on resource-constrained embedded systems.

8.1 Efficient Inference of BNNs

Achieving efficiency is a central challenge for both standard neural networks and BNNs. This section reviews related efforts, beginning with compiler and compression techniques for deterministic models on embedded hardware, and then focusing on methods that aim to make BNNs feasible under similar constraints.

Resource-Efficient Deployment on Mobile Devices. Deploying machine learning models on mobile hardware requires strict control of computational and memory budgets. To this end, *deep learning compilers* have emerged as a key abstraction layer that bridges high-level machine learning frameworks and heterogeneous hardware backends. They automate optimizations such as operator fusion, memory scheduling, and parallelization, and increasingly support automated tuning of implementation schedules. Prominent examples include Tensor Virtual Machine (TVM) [115] and Multi-Level Intermediate Representation (MLIR) [45], which provide extensible infrastructures for graph- and operator-level optimizations across CPUs, GPUs, and reconfigurable hardware such as FPGAs. TVM in particular combines these optimizations with learning-based auto-tuning, enabling efficient code generation across diverse backends. We build on this abstraction in the present chapter by extending TVM with custom operators to support the non-standard operators of the Probabilistic Forward Pass.

Model compression techniques such as pruning and quantization reduce the computational and memory footprint of neural networks, enabling deployment on embedded hardware [120], [133], [178]. More advanced approaches use reinforcement learning or neural architecture search to automatically adapt sparsity and precision across layers [21], [90]. An introduction to compression methods and their hardware implications has already been provided in Chapter 3, while Galen (Chapter 4) extends these ideas with automatic compression guided by sensitivity analysis and hardware measurements. For a broader survey of efficient neural network (NN) inference, we refer to Roth [12].

From a Bayesian perspective, Louizos, Ullrich, and Welling proposed compression techniques that embed priors into pruning and use posterior uncertainty to assign precision [160]. These methods, however, are designed for deterministic networks and do not address efficiency in BNNs.

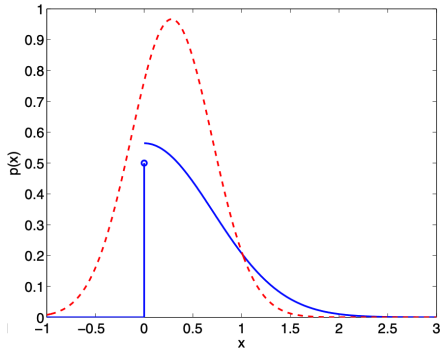


Figure 8.1 Moment matching in PFP, shown for the ReLU activation. A Gaussian input distribution truncated by the non-linearity (solid) is approximated by a Gaussian (dashed) with matching first and second moment.

Bayesian Neural Networks on Resource-Constrained Devices. Lightweight BNN approximations such as Monte Carlo Dropout (MCDO) [175] and Deep Ensembles [154] are widely used in practice. They reduce training cost compared to full Bayesian inference, but sacrifice theoretical rigor and, at inference time, still require multiple forward passes. Although significantly more efficient than sampling-based BNNs, SVI remains too costly for embedded systems, where latency and energy budgets are highly constrained.

Efforts that more directly target BNNs on constrained hardware remain limited. Banerjee, Kalbarczyk, and Iyer, for example, introduced AcMC2, a compiler that maps probabilistic models to optimized MCMC-based accelerators, implemented either as ASICs or on FPGAs [81]. However, the focus is on general probabilistic models rather than BNNs. Other dedicated accelerators, including ShiftBNN [51], B^2N^2 [14], and VIBNN [113], use FPGAs or ASICs together with large-scale Gaussian random number generators to accelerate BNNs.

While methods like MCDO and ensembles provide practical workarounds, their reliance on repeated forward passes limits applicability on embedded devices. No existing work demonstrates efficient deployment of SVI-based BNNs on such platforms.

8.2 Probabilistic Forward Pass

The concept of the Probabilistic Forward Pass was first introduced by Roth [48], [186] as an extreme approximation of SVI-based BNNs. By extending the Gaussian assumption from weights to activations, Roth showed that distributions

can be propagated in closed form throughout the network, replacing repeated sampling with deterministic transformations. A key idea is *moment matching*: whenever a non-Gaussian distribution arises within the network, it is mapped back to a Gaussian by matching the first two moments. This ensures that propagation remains tractable while still capturing predictive uncertainty. Figure 8.1 illustrates this principle for the widely used ReLU activation, where a Gaussian activation is truncated by the non-linearity and then transformed back into Gaussian form.

Consider a fully connected layer with Gaussian inputs characterized by mean and variance $(\mu_{\text{in}}, \sigma^2_{\text{in}})$ and Gaussian weights with parameters (μ_w, σ^2_w). The output distribution is obtained by propagating expectations and variances through the linear transformation. The mean is computed by propagating expectations, while the variance reflects uncertainty from both weights and inputs. PFP assumes independence between activations—a mean-field approximation that makes variance propagation tractable. Equations 8.1 and 8.2 show the scalar form for the mean and variance of neuron i in layer l:

$$\mu_{a_i^l} = \sum_{j=1}^{d_{l-1}} \mu_{w_{ij}^l} \cdot \mu_{x_j^{l-1}} \tag{8.1}$$

$$\sigma^2_{a_i^l} = \sum_{j=1}^{d_{l-1}} \sigma^2_{w_{ij}^l} \cdot \mathbb{E}\left[\left(x_j^{l-1}\right)^2\right] + \mu^2_{w_{ij}^l} \cdot \left(\mathbb{E}\left[\left(x_j^{l-1}\right)^2\right] - \mu^2_{x_j^{l-1}}\right). \tag{8.2}$$

Here the *second raw moment* $\mathbb{E}(x^2) = \mu^2 + \sigma^2$ is applied. The variable d_{l-1} denotes the width of the previous layer, i.e., the input dimension to layer l. This can also be written in vectorized form using mean and variance directly:

$$\boldsymbol{\sigma}_{\boldsymbol{a}}^{\mathbf{2}} = \boldsymbol{\sigma}_{\boldsymbol{w}}^2 \cdot \mathbb{E}[\boldsymbol{x}^2] + \boldsymbol{\mu}_{\boldsymbol{w}}^2 \cdot \left(\mathbb{E}[\boldsymbol{x}^2] - \boldsymbol{\mu}_{\boldsymbol{x}}^2\right) \tag{8.3}$$

$$= \boldsymbol{\sigma}_{\boldsymbol{w}}^2 \cdot \boldsymbol{\mu}_{\boldsymbol{x}}^2 + \boldsymbol{\mu}_{\boldsymbol{w}}^2 \cdot \boldsymbol{\sigma}_{\boldsymbol{x}}^2 + \boldsymbol{\sigma}_{\boldsymbol{w}}^2 \cdot \boldsymbol{\sigma}_{\boldsymbol{x}}^2 \tag{8.4}$$

Later implementations support both formulations and select the computationally cheaper one. To avoid costly conversions, the outputs of one layer and inputs of the next are kept in a consistent form, either as mean–variance pairs or as mean–second-raw-moment pairs.

A key challenge arises in non-linear activation functions, where Gaussianity

is not preserved. For the widely used ReLU, PFP therefore applies the moment-matching procedure already introduced above (cf. Figure 8.1). The truncated Gaussian produced by the non-linearity is projected back into Gaussian form by matching its first and second moments, keeping propagation in closed form. The resulting expressions for mean and second raw moment are:

$$\mu_{x_i^l} = \mathbb{E}[x_i^l] = \frac{\mu_{a_i^l}}{2}\left(1 + \operatorname{erf}\left(\frac{\mu_{a_i^l}}{\sqrt{2\sigma^2_{a_i^l}}}\right)\right) + \sqrt{\frac{\sigma^2_{a_i^l}}{2\pi}}\exp\left(-\frac{\mu^2_{a_i^l}}{2\sigma^2_{a_i^l}}\right) \tag{8.5}$$

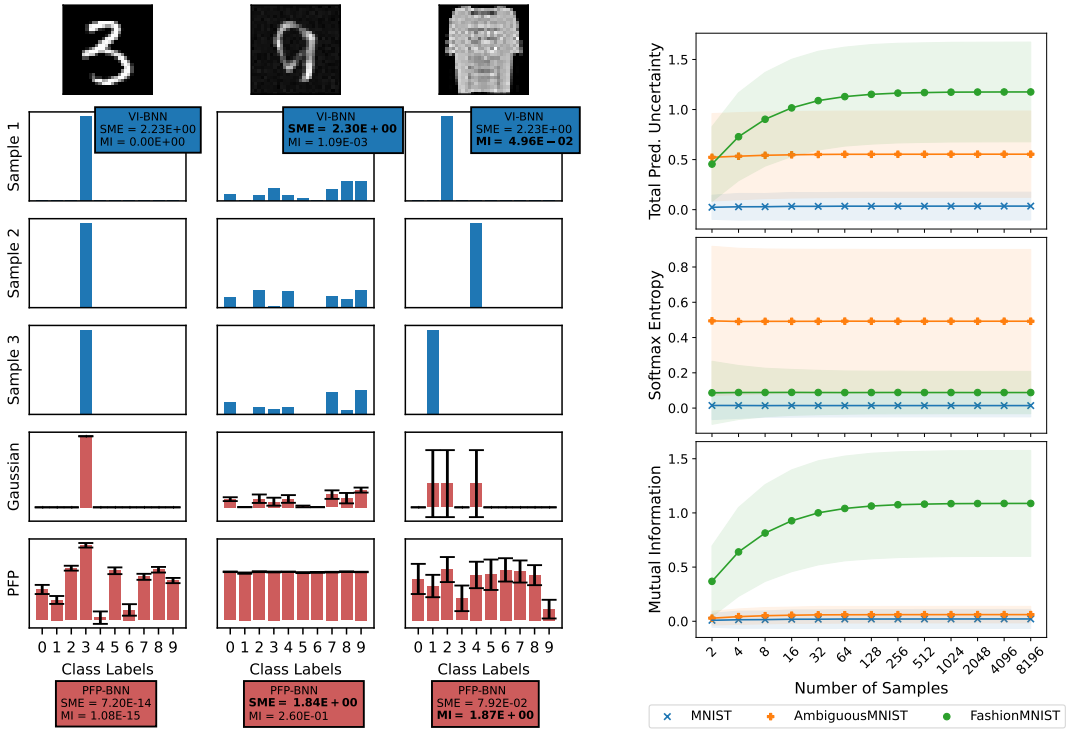
$$\mathbb{E}\left[(x_i^l)^2\right] = \frac{\sigma^2_{a_i^l} + \mu^2_{a_i^l}}{2}\left(1 + \operatorname{erf}\left(\frac{\mu_{a_i^l}}{\sqrt{2\sigma^2_{a_i^l}}}\right)\right) + \mu_{a_i^l}\sqrt{\frac{\sigma^2_{a_i^l}}{2\pi}}\exp\left(-\frac{\mu^2_{a_i^l}}{2\sigma^2_{a_i^l}}\right). \tag{8.6}$$

Here $\operatorname{erf}(u) = \frac{2}{\sqrt{\pi}}\int_0^u e^{-z^2}dz$ denotes the error function [48]. Propagating distributions through successive layers in this way yields the final predictive distribution at the network output. During inference, predictions require only a single pass, directly producing expected outputs and uncertainties without ensembles or repeated runs. By reformulating the computation to operate in closed form on distributions, PFP provides a scalable and efficient pathway to deploy BNNs in practice.

To illustrate the effect of predictive sampling and the advantages of PFP, Figure 8.2 compares uncertainty estimation in an SVI-based BNN and in PFP. Panel a shows predictions on three example images from MNIST [232], Ambiguous-MNIST [31], and Fashion-MNIST [169], the latter serving as an out-of-distribution example.

Recall that softmax entropy (SME) quantifies *aleatoric uncertainty* as the average class entropy across predictive samples. It reflects variability already present within individual predictions, but is aggregated by averaging over all samples. In contrast, mutual information (MI) measures *epistemic uncertainty* as the disagreement between predictive samples, and therefore requires variability across samples to become visible. Panel b illustrates this for SVI: while SME stabilizes quickly, reliable MI estimates—critical for OOD detection—converge only after many samples, making SVI computationally demanding.

PFP, by contrast, propagates means and variances jointly in closed form. This eliminates the need for repeated sampling and produces both uncertainty



(a) Illustration of BNN Predictions

(b) Effect of Number of Samples in SVI-BNNs

Figure 8.2 Comparison of uncertainty estimation in SVI-based BNNs and PFP. (a) Exemplary predictions on MNIST [232], Ambiguous-MNIST [31], and Fashion-MNIST [169] as OOD sample. For each dataset, predictions are shown for an SVI-based BNN (blue samples), its Gaussian approximation, and PFP. Variability across class probabilities captures aleatoric uncertainty (SME), while disagreement between samples indicates epistemic uncertainty (MI). (b) Influence of the number of SVI predictive samples on uncertainty metrics, showing that reliable mutual information estimates require many samples, whereas PFP obtains them in a single pass.

measures in a single forward pass. While this comes at the cost of a less sharp separation between aleatoric and epistemic components, it drastically reduces computational overhead compared to sampling-based inference.

8.2.1 Conceptual Limitations

The efficiency of PFP comes at the price of assuming Gaussian-distributed logits. This structural simplification ensures closed-form propagation, but it limits the ability to represent non-Gaussian predictive distributions.

As a result, total predictive uncertainty—quantified by Shannon Entropy—is

Table 8.1 Influence of Gaussian approximation on uncertainty metrics. While total uncertainty remains consistent, the separation of aleatoric and epistemic components may degrade under strong epistemic uncertainty.

Unc. Regime	Total		Softmax Entropy		Mutual Information	
	True	Gauss	True	Gauss	True	Gauss
Low	1.5008	1.5007	1.5007	1.5007	0.0	0.0
Aleatoric high	1.6094	1.6094	1.6094	1.6094	0.0	0.0
Epistemic high	1.6094	1.6094	1.5002	1.5484	**0.1092**	**0.0610**

largely preserved, but the decomposition into SME (aleatoric) and MI (epistemic) can become biased. Table 8.1 illustrates this effect using artificial regimes with low, high-aleatoric, and high-epistemic uncertainty. While the Gaussian approximation faithfully reproduces total uncertainty in all cases, errors arise under high epistemic uncertainty, where mutual information is underestimated by about 44%.

In practice, this means that PFP preserves overall uncertainty levels, but may compromise the disentanglement of aleatoric and epistemic components whenever predictive distributions deviate substantially from Gaussianity.

8.3 PFP Training and Uncertainty Estimation

A central advantage of the Probabilistic Forward Pass is its compatibility with pretrained SVI models. It benefits from the relatively fast training of SVI while leveraging established tools for constructing SVI-based BNNs. Probabilistic programming languages such as Pyro [82], Stan,[1] and TensorFlow Probability[2] provide flexible frameworks for designing, training, and evaluating probabilistic models. Here, BNNs are trained with Pyro SVI and then exported for use with PFP.

Two neural architectures are considered in the experiments. First, a simple MLP with a single hidden layer of 100 neurons. Second, the LeNet-5 [232] architecture. In both cases, all weights are treated probabilistically with Gaussian priors. The mean-field assumption [61] is applied to simplify training by neglecting correlations between Gaussian weight distributions.

[1]https://mc-stan.org/
[2]https://github.com/tensorflow/probability

Training with SVI resembles standard neural network optimization but introduces additional loss terms and hyperparameter sensitivities. Due to the multi-objective nature of the evidence lower bound, training is slower and requires careful initialization. In our setup, SVI-BNNs are trained for 1000 epochs using the Adam optimizer [207] with a fixed learning rate of 0.001. Variational posterior weights are initialized with $\mu = 0.08$ and $\sigma = 0.0001$, and a mini-batch size of 100 is used for training.

Balancing the expected log-likelihood with the KL divergence term in the ELBO is non-trivial. Instead of a fixed scaling factor, we employ *KL annealing* [108], [204]. Here, the KL term weight $A(e)$ is gradually increased from 0 to $\alpha_{\max} = 0.25$ over the training epochs e,

$$\mathrm{ELBO}(q, e) = \mathbb{E}[\log p(\mathcal{D}|w)] - A(e) \cdot \mathrm{KL}(q(w)||p(w|\mathcal{D})). \tag{8.7}$$

This strategy improves robustness to initialization and avoids the need for non-probabilistic pretraining.

The trained means and variances of the weights can be directly consumed by PFP. A conversion from logarithmic to normal parameterization is performed, followed by an uncertainty calibration step. This calibration applies a global scaling to the variances and is referred to as the *calibration factor*.

We hypothesize that the need for calibration arises because moment-based propagation only tracks the first two moments of the distributions and ignores higher-order terms. At every non-linear transformation, the resulting non-Gaussian distribution is projected back to Gaussian form, which introduces systematic mismatches compared to full sampling. We assume that the main contribution to this miscalibration stems from such moment-matching errors, where higher-order moments beyond the mean and variance are neglected. In addition, approximations in the original derivation of PFP, such as the use of first-order Taylor expansions for nonlinearities [186], may further contribute to the discrepancy. These effects accumulate across network components, and the calibration factor provides a pragmatic global correction that aligns the propagated variances with the uncertainties observed under sampling-based inference.

To evaluate the ability of BNNs to capture both aleatoric and epistemic uncertainty, we use the Dirty-MNIST dataset introduced by Mukhoti, Kirsch,

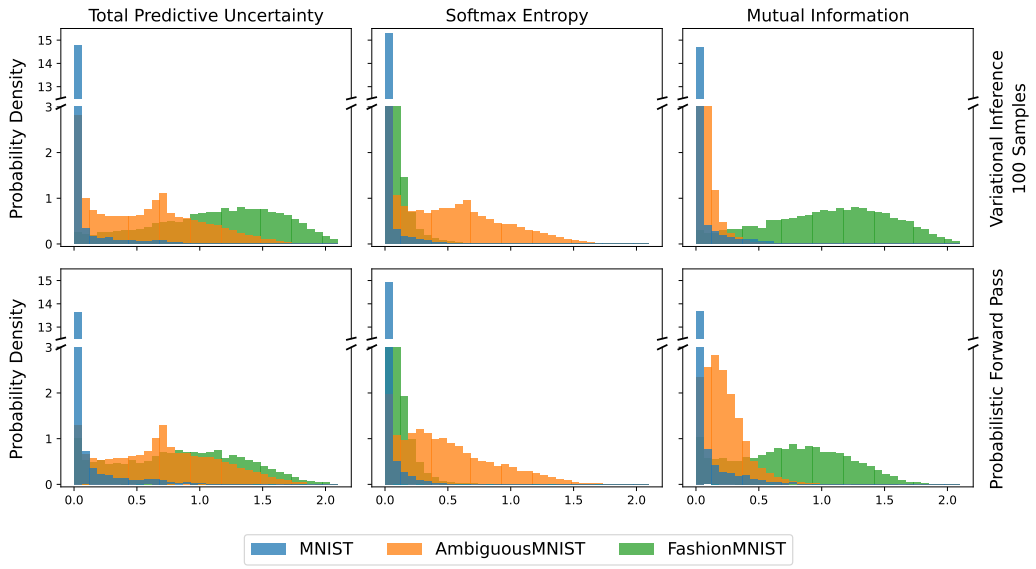


Figure 8.3 Comparison of uncertainty predictions obtained with SVI and PFP. On MNIST [232], both approaches yield low uncertainty, as expected for in-domain data. Ambiguous-MNIST [31] exhibits higher aleatoric uncertainty, quantified by softmax entropy. Fashion-MNIST [169], used here as an out-of-distribution dataset, shows increased epistemic uncertainty, reflected in higher mutual information. Overall, both methods correctly assign most samples to their expected domains.

Amersfoort, Torr, and Gal [31] as presented in Chapter 7.4.

8.3.1 Qualitative Performance

To assess the approximation quality of PFP, we compare its predictions to sampling-based SVI across the Dirty-MNIST dataset. Figure 8.2a shows example predictions, while Figure 8.3 reports the aggregated metrics Shannon Entropy, softmax entropy, and mutual information. As illustrated in Figure 8.2b, the number of samples strongly influences Shannon Entropy and, consequently, mutual information. Unlike SVI, PFP does not provide an explicit sampling dimension. To ensure comparability, we introduce an artificial sampling procedure based on the PFP-predicted logit means μ_{PFP} and variances σ^2_{PFP}. Synthetic samples l_{PFP} are generated as

$$l_{\text{PFP}} \sim \mathcal{N}(\mu_{\text{PFP}}, \sigma^2_{\text{PFP}}). \tag{8.8}$$

Table 8.2 Comparison of SVI and PFP-based BNNs on Dirty-MNIST [31].

Metric	Method	MLP	LeNet-5
Calibration Factor	PFP	0.3	0.4
Accuracy	SVI	96.3%	98.7%
	PFP	96.3%	98.9%
AUROC	SVI	0.812	0.986
	PFP	0.858	0.966

This *logit sampling* is computationally lightweight, avoiding repeated forward passes. It enables the calculation of uncertainty metrics used in sampling-based methods. In highly constrained applications, the raw PFP-predicted variances can be directly employed for decision making.

Figure 8.3 shows that both SVI and PFP report elevated uncertainty for Ambiguous-MNIST and Fashion-MNIST, compared to MNIST. Both methods capture the expected patterns: increased softmax entropy for ambiguous inputs and increased mutual information for OOD inputs. Overall, the predictions are consistent with theoretical expectations.

A more detailed analysis is presented in Figure 8.4, which plots softmax entropy against mutual information across all samples. Here, SVI achieves cleaner separation of aleatoric and epistemic uncertainty. Nevertheless, PFP provides a practically sufficient distinction in most cases. In rare edge cases both measures take high values, reducing separability.

Table 8.2 compares the two methods on Dirty-MNIST. Both achieve similar predictive accuracy and AUROC, with PFP requiring only a single forward pass. The influence of network architecture is visible, as the convolutional model achieves better accuracy and separation than the MLP. In summary, PFP delivers uncertainty estimates close to SVI while offering significantly improved efficiency.

8.4 PFP Operator Library with TVM

TVM provides multiple internal languages and intermediate representations, including TensorIR [17], tensor expression (TE), TVMScript, and Relax [22]. These abstractions are essential for implementing custom operators. tensor expressions define computation rules in a compact way, while TensorIR organizes computations as modular *blocks*, enabling fine-grained scheduling and optimizations. Relax [22], the successor of Relay [132], serves as a high-level intermediate

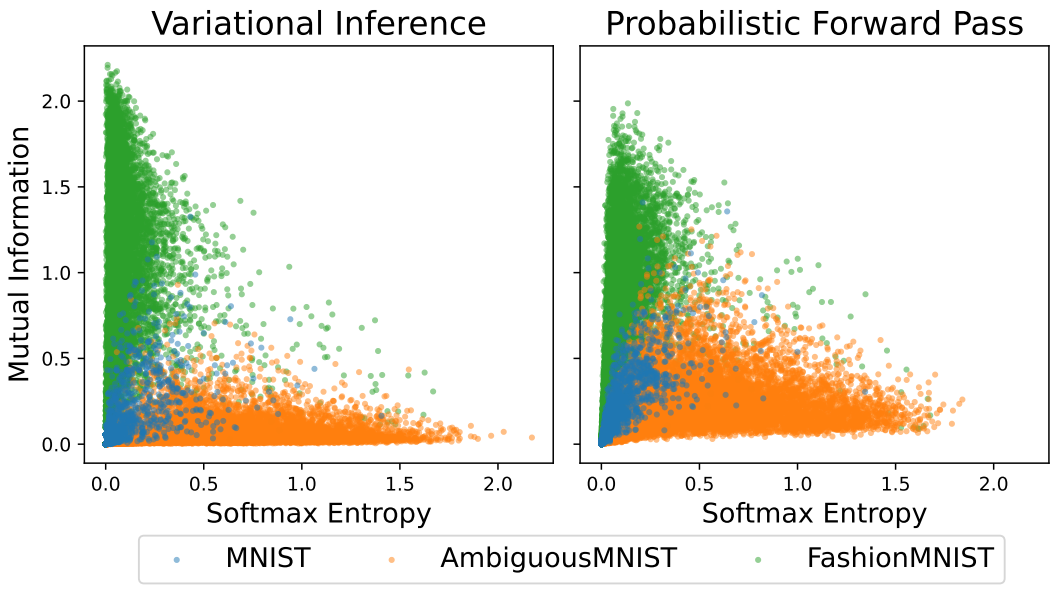


Figure 8.4 Comparison of aleatoric and epistemic uncertainty estimates obtained with SVI and PFP for a LeNet-5 BNN. The scatter plot displays softmax entropy (aleatoric uncertainty) versus mutual information (epistemic uncertainty). While SVI provides a clearer disentanglement of the two uncertainty types, PFP has some overlap, but still achieves a practically useful degree of separation.

representation, supporting dynamic shapes, control flow, and integration with TensorIR.[3] TVMScript, a Python-based frontend, allows direct definition and modification of TensorIR and Relax programs.

To implement a custom operator, developers specify the computation in TE and generate IRModules via the BlockBuilder API. BlockBuilder creates primitive functions from TE expressions, which are then connected through Relax and optimized with TensorIR scheduling.[4] This workflow enables efficient execution of specialized operators across diverse hardware platforms while minimizing implementation complexity.

Operating on Tuples

Neural network operators frequently take multiple input tensors, but operators producing multiple outputs are relatively uncommon. PFP introduces a particular requirement, as both mean and variance must be propagated through the network. TVM follows the principle of *one operator = one compute rule*, meaning each

[3]https://tvm.apache.org/docs/deep_dive/relax/learning.html
[4]https://mlc.ai/docs/get_started/tutorials/quick_start.html

operator executes a single sequential computation without divergence. As a result, PFP operations may be split into separate operators, e.g., one for the mean and one for the variance. However, this design increases interconnection overhead, complicates network construction, and prevents reuse of shared subterms between the mean and variance paths. A joint formulation, in which both quantities are computed together, allows reuse of intermediate results and avoids redundant computations. Figure 8.5 shows that such joint operators consistently outperform separate implementations by improving data reuse and memory locality.

Variance and Second Raw Moment

The original formulation of PFP operators is based on mean and variance inputs and outputs. However, reformulating Equation 8.2 in terms of second raw moments improves reuse and reduces conversions. In this form, the variance of activations in a dense layer becomes

$$\sigma^2_{a_i^l} = \sum_{j=1}^{d_{l-1}} \mathbb{E}\left[\left(w_{ij}^l\right)^2\right] \cdot \mathbb{E}\left[\left(x_j^{l-1}\right)^2\right] - \left(\mu_{w_{ij}^l} \cdot \mu_{x_j^{l-1}}\right)^2, \tag{8.9}$$

where the pre-computed second raw moments of the weights and activations can be directly reused. This eliminates conversions from activation outputs, which already produce second raw moments by design. The resulting tuple-based operator is more cache-efficient and reduces overall runtime. Figure 8.5 illustrates the performance benefits of the second raw moment formulation and joint operators.

When consecutive layers differ in their representation of variances and second raw moments, conversion is straightforward using $\mathbb{E}(x^2) = \mu^2 + \sigma^2$. However, performing repeated conversions across layers is wasteful. To address this, the operator implementation provides a configurable conversion function, while ensuring consistency between layers remains the responsibility of the model designer. Weights must also follow this convention, being stored either as means and variances (see Equation 8.4) or as means and second raw moments (see Equation 8.2).

By default, compute layers such as dense and convolutional layers expect second raw moments as inputs and produce variances as outputs. Conversely,

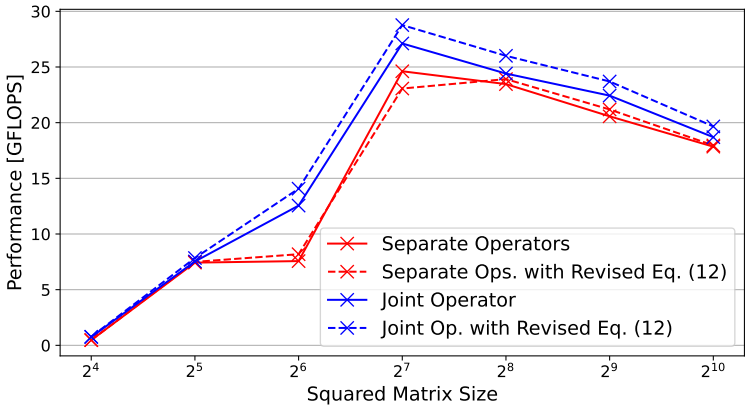


Figure 8.5 Comparison of operator implementations on an ARM Cortex-A72. The results demonstrate the performance gains from reformulating Equation 8.2 into Equation 8.9, and from employing joint operators that combine mean and variance paths instead of implementing them separately.

activation functions take variances as inputs and produce second raw moments. This convention ensures compatibility between compute and activation functions operators. Additional layers, such as Max Pooling, which consume and produce variances, require explicit conversion at their interfaces.

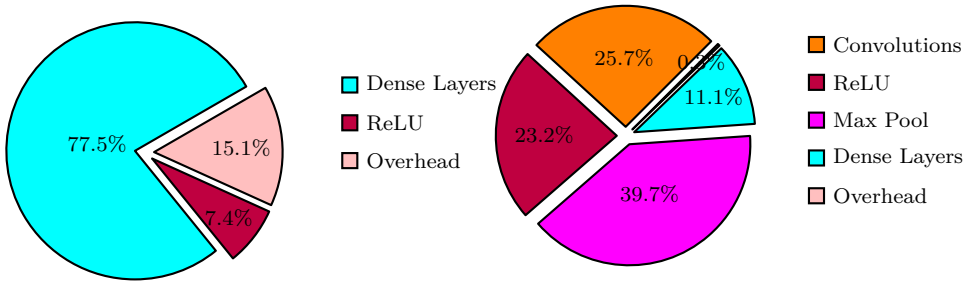
A special case occurs in the first network layer, where no input variance is available. In this case, the forward propagation simplifies to

$$\mu_{a_i^l} = \sum_{j=1}^{d_{l-1}} \mu_{w_{ij}^l} \cdot \mu_{x_j^{l-1}}, \tag{8.10}$$

$$\sigma^2_{a_i^l} = \sum_{j=1}^{d_{l-1}} \sigma^2_{w_{ij}^l} \cdot \mu^2_{x_j^{l-1}}. \tag{8.11}$$

Here, weight variances are required explicitly. For subsequent layers, storing weights as second raw moments avoids additional conversions. Furthermore, compute layers support three bias configurations: no bias, deterministic bias, and probabilistic bias with variances.

In summary, integrating custom PFP operators into TVM requires specific design considerations. Our analysis shows that combining joint operator implementations with second raw moment formulations yields the best efficiency across tested architectures.



(a) MLP (b) LeNet-5

Figure 8.6 Distribution of execution time across operator types for PFP-based BNNs, measured on a Cortex-A72 with a mini-batch size of 10. For the MLP, dense layers dominate runtime. For LeNet-5, latency is more evenly distributed, and operators considered simple in deterministic settings, such as ReLU and Max Pooling, contribute significant overhead

8.5 Optimizing for Performance

The PFP implementation introduced in the previous Section already enables functional BNNs with uncertainty estimation at a fraction of the cost of sampling-based methods. Nevertheless, further performance can be gained through implementation-level and hardware-aware optimizations. We first profile operator costs to identify performance bottlenecks. Based on these insights, we apply manual optimizations tailored to dense and pooling operators. Subsequently, we evaluate the effectiveness of automatic tuning frameworks provided by the TVM compiler. Finally, we benchmark the tuned implementations across different CPUs and compare them against SVI-based BNNs.

8.5.1 Profiling Operators

Optimizing for performance requires identifying operators that dominate execution time. TVM provides three execution modes for compiled binaries: standard execution, benchmarking with averaging for high precision, and profiling, which reports latency on a per-operator basis. These profiling capabilities enable both quantitative evaluation of optimization strategies and visualization of operator cost distribution.

Figure 8.6 shows the runtime share of different operator types. For the MLP, dense layers are the dominant cost. For LeNet-5, latency is more balanced, with ReLU and Max Pool operators accounting for a substantial portion of

Table 8.3 Evaluation of manual optimization techniques for the PFP dense operator. Measurements were performed on a Cortex-A72 using a 3-layer MLP with mini-batch size 10.

Optimizations		Latency		Speedup
Name	**Other Opt.**	**without Opt.**	**with Opt.**	
Baseline (no tuning)	OFF	3.760 ms	-	-
Baseline (min. tuning)	OFF	3.681 ms	-	-
Tiling[1]	OFF	3.672 ms	0.747 ms	4.91×
Loop Reordering	OFF	3.681 ms	1.940 ms	1.90×
Vectorization	OFF	3.681 ms	8.837 ms	0.42×
Parallelization	OFF	3.681 ms	0.729 ms	5.05×
Loop Unrolling	OFF	3.681 ms	1.967 ms	1.87×
Tiling[1]	ON	0.754 ms	4.237 ms	0.18×
Loop Reordering	ON[2]	0.750 ms	0.743 ms	1.01×
Vectorization	ON[2]	0.759 ms	0.743 ms	1.02×
Parallelization	ON[2]	1.953 ms	0.743 ms	2.63×
Loop Unrolling	ON[2]	3.042 ms	0.743 ms	4.09×
All Optimizations	ON[2]	3.760 ms	0.743 ms	5.06×

[1] Without stochastic tuning.
[2] All optimizations in use except tiling.

total runtime. This highlights that seemingly simple operators can become computationally expensive when propagating distributions rather than scalars.

8.5.2 Manual Optimizations

The profiling results indicate that dense layers are the primary bottleneck in the MLP. We therefore target the dense operator with optimization techniques commonly used for matrix-matrix multiplication. These include tiling, loop reordering, loop unrolling, vectorization, and parallelization.

Table 8.3 shows the effect of individual and combined optimizations. Some techniques, such as vectorization, degrade performance when used in isolation, as they require loop structures to be reordered first. Loop unrolling and parallelization are the most effective optimizations for the PFP dense operator. When combined, they yield a speedup of more than 5×.

Tiling requires separate evaluation. Applied independently with hand-tuned tile sizes, it yields strong performance gains. However, tiling is the only optimization that does not support stochastic tuning. Since the other optimizations benefit considerably from stochastic tuning, enabling tiling disables this option. Consequently, applying all optimizations including tiling but without stochas-

Table 8.4 Evaluation of Max Pool implementations for LeNet-5 on a Cortex-A72 with mini-batch size 10. The specialized vectorized operator outperforms the generic reduction-based implementation, while automatic schedules fail to improve performance.

Arch.	Implementation	Auto-tuning	Latency	
			Max Pools	**Entire NN**
LeNet-5	Generic Max Pool	No	12.09 ms	29.13 ms
LeNet-5	Generic Max Pool	All operators	5.04 ms	10.74 ms
LeNet-5	Generic Max Pool	All except Max Pool	11.92 ms	17.82 ms
LeNet-5	Vect. Max Pool	No	3.54 ms	21.10 ms
LeNet-5	Vect. Max Pool	All operators	27.28 ms	33.42 ms
LeNet-5	Vect. Max Pool	All except Max Pool	3.69 ms	9.79 ms

tic tuning performs worse than tiling alone. The best results are obtained by combining all other optimizations with stochastic tuning while excluding tiling.

Max Pool Operator For LeNet-5, the generic Max Pooling operator introduced by [48], implemented as a reduction, proved inefficient. To address this, we implemented a specialized vectorized variant with fixed kernel size $k = 2$. As shown in Table 8.4, automatic schedules fail to improve this operator and in some cases even degrade performance. Consequently, the custom Max Pool implementation is excluded from automatic tuning.

8.5.3 Automatic Optimizations

Beyond manual schedules, TVM provides advanced automatic tuning frameworks [26], [32], [79], [115]. The *Meta Scheduler* [32] automatically explores large optimization spaces by generating and benchmarking candidate schedules. This approach is slower than expert-crafted tuning but typically achieves comparable performance and requires no manual effort. Applying Meta Scheduler to the PFP dense operator achieves latencies nearly identical to hand-tuned schedules (0.742 ms versus 0.743 ms). We therefore rely on it in subsequent experiments.

Table 8.5 summarizes profiling results for the MLP and LeNet-5 before and after tuning. Dense and convolution layers benefit substantially from optimization, delivering the largest performance improvements.

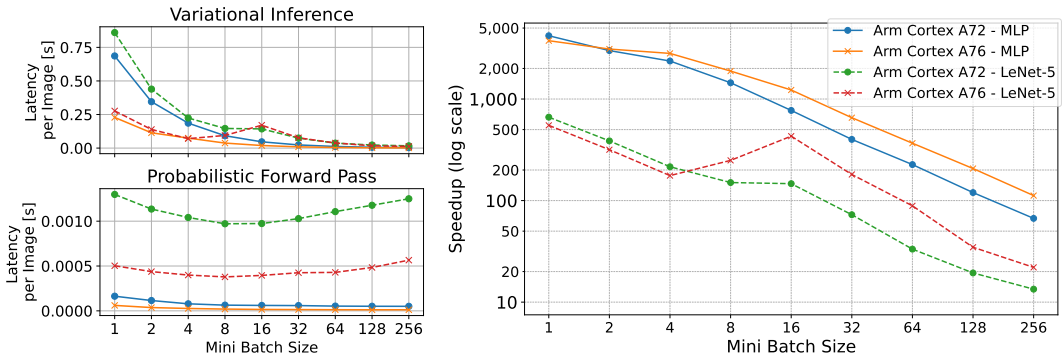
Table 8.5 Profiling of PFP-based neural architectures on a Cortex-A72 with mini-batch size 10. The largest performance gains after tuning are achieved in dense and convolution layers.

Arch.	Layer	Baseline		Tuned Impl.		Speedup
		Latency	Fraction	Latency	Fraction	
MLP	Dense 1	2.931 ms	62.8 %	0.642 ms	33.7 %	4.6×
	Dense 2	0.570 ms	12.2 %	0.125 ms	6.6 %	4.6×
	ReLU[1]	0.195 ms	4.2 %	0.185 ms	9.7 %	1.1×
	Dense 3	0.063 ms	1.3 %	0.036 ms	1.9 %	1.8×
	Sum	3.846 ms	82.4 %	1.078 ms	56.5 %	3.6×
	Entire Model	4.668 ms	100.0 %	1.908 ms	100.0 %	2.4×
LeNet-5	Conv2d 2	7.207 ms	30.4 %	1.509 ms	12.4 %	4.8×
	ReLU 1	4.133 ms	17.4 %	2.153 ms	17.7 %	1.9×
	Dense 1	2.791 ms	11.8 %	0.516 ms	4.2 %	5.4×
	Max Pool 1[1,2]	2.780 ms	11.7 %	2.807 ms	23.1 %	1.0×
	ReLU 2	1.541 ms	6.5 %	0.980 ms	8.1 %	1.6×
	Conv2d 1	0.949 ms	4.0 %	0.552 ms	4.5 %	1.7×
	Max Pool 2[2]	0.854 ms	3.6 %	0.902 ms	7.4 %	0.9×
	Dense 2	0.589 ms	2.5 %	0.111 ms	0.9 %	5.3×
	ReLU 3	0.168 ms	0.7 %	0.071 ms	0.6 %	2.4×
	ReLU 4	0.107 ms	0.5 %	0.051 ms	0.4 %	2.1×
	Dense 3	0.057 ms	0.2 %	0.027 ms	0.2 %	2.1×
	Sum	21.751 ms	91.8 %	10.226 ms	84.1 %	2.1×
	Entire Model	23.698 ms	100.0 %	12.166 ms	100.0 %	1.9×

[1] Layers present multiple times in network
[2] Layers excluded from tuning

Table 8.6 Latency and speedup comparison of deterministic, SVI, and PFP-based networks on three different embedded ARM processors, using vectorized Max Pool. While PFP incurs some overhead compared to deterministic inference, it achieves orders-of-magnitude speedups over sampling-based SVI, making uncertainty-aware inference practical on embedded devices.

Arch.	BS	Processor	Deterministic NN		SVI	PFP		Speedup
			not tuned	tuned		not tuned	tuned	
MLP	10	Cortex-A53	14.02 ms	0.933 ms	-	15.26 ms	4.989 ms	
		Cortex-A72	4.59 ms	0.186 ms	734.74 ms	3.75 ms	0.742 ms	990.2×
		Cortex-A76	1.64 ms	0.071 ms	307.52 ms	1.89 ms	0.341 ms	901.8×
MLP	100	Cortex-A53	137.80 ms	6.565 ms	-	147.61 ms	15.358 ms	
		Cortex-A72	45.81 ms	1.134 ms	775.32 ms	36.33 ms	5.182 ms	149.6×
		Cortex-A76	16.30 ms	0.230 ms	306.89 ms	18.60 ms	1.200 ms	255.7×
LeNet-5	10	Cortex-A53	21.14 ms	4.726 ms	-	76.09 ms	35.159 ms	
		Cortex-A72	6.89 ms	0.754 ms	1196.42 ms	21.23 ms	10.022 ms	119.4×
		Cortex-A76	3.16 ms	0.347 ms	801.40 ms	9.63 ms	3.897 ms	205.6×
LeNet-5	100	Cortex-A53	209.28 ms	41.697 ms	-	801.33 ms	383.680 ms	
		Cortex-A72	70.08 ms	9.524 ms	2708.16 ms	240.94 ms	116.330 ms	23.3×
		Cortex-A76	31.51 ms	3.131 ms	2488.73 ms	119.76 ms	45.039 ms	55.3×



(a) Latency per image (b) Relative speedup of PFP over SVI

Figure 8.7 Latency and speedup in relation to mini-batch size. SVI-based BNNs, evaluated with 30 samples, show high per-image latency and poor scalability at small batch sizes. In contrast, PFP maintains stable latency across all sizes due to targeted tuning. As a result, speedups range from tens to over four thousand, with the strongest gains appearing for batch size 1, which is a common case for real-time embedded applications.

8.5.4 Evaluation and Performance Gain

Table 8.6 and Figure 8.7 highlight the central result of this chapter: PFP, combined with compiler-based optimizations, reduces inference cost by several orders of magnitude compared to sampling-based SVI. Across ARM processors, speedups reach on average 574× for the MLP and 101× for LeNet-5, despite PFP requiring more complex operators and doubling both parameters and activations relative to deterministic inference.

The importance of these gains becomes most evident under conditions typical for embedded systems, where real-time operation often requires small mini-batch sizes. In this regime, SVI-based BNNs, implemented in Pyro with 30 samples—already a minimal configuration for meaningful uncertainty estimation (see also Figure 8.2b)—still incur substantial latency. Figure 8.7a shows that their runtime scales poorly as batch sizes decrease. By contrast, PFP benefits directly from compiler support: it is recompiled and optimized for each mini-batch size individually, allowing it to maintain stable latency across configurations. Only minor fluctuations remain, which can be attributed to cache effects or SIMD alignment. The resulting speedups, shown in Figure 8.7b, range from 13× to 112× at batch size 256, and increase dramatically to between 550× and 4200× for batch size 1—a common real-time scenario.

Altogether, these experiments demonstrate that the combination of prob-

abilistic approximation and compiler-based code generation makes efficient, uncertainty-aware inference feasible on resource-constrained embedded CPUs—a capability that was previously out of reach for SVI-based BNNs.

Summary

Although Bayesian neural networks offer a principled framework for uncertainty estimation, their deployment on embedded systems remains constrained by the computational demands of sampling-based inference. This chapter demonstrated how combining algorithmic approximation with compiler-level optimization can overcome these limitations. By replacing repeated stochastic evaluations with a single closed-form forward pass, the Probabilistic Forward Pass enables efficient execution of SVI-trained networks on embedded ARM CPUs. While the Gaussian assumption limits expressiveness, it yields substantial speedups whenever the approximation remains adequate.

To realize this approach in practice, we extended the deep learning compiler TVM with specialized probabilistic operators and applied both targeted scheduling and automatic tuning. The resulting implementation achieved speedups of up to 4200× on ARM processors while preserving accuracy and reliable uncertainty estimation under tight resource constraints. Together, these contributions establish a practical pathway from Bayesian approximation to deployment, making probabilistic inference feasible on embedded hardware.

While PFP provides an analytic route to efficient uncertainty estimation, the next chapter explores a complementary strategy based on ensemble methods. These approaches trade analytical simplicity for representational flexibility, capturing multiple modes of the predictive distribution with high parallel efficiency.

9

Ensemble Methods for Practical Bayesian Neural Networks

The best way to have a good idea is to have a lot of ideas.

— Linus Pauling

Classical Bayesian neural network inference techniques such as MCMC or SVI provide high quality posterior approximations and are mathematically grounded, but their computational cost scales poorly with model size [23]. As shown in Chapter 7, the high dimensionality of modern neural networks renders exact or near-exact Bayesian inference methods impractical for most real-world tasks. This computational bottleneck is especially pronounced in resource-constrained environments such as embedded devices, where memory and energy budgets are tightly limited [12].

A practical alternative is to approximate the posterior through ensembles of predictors rather than through explicit weight distributions. In this paradigm, multiple predictors are trained or sampled, and their collective variability is used as a proxy for epistemic uncertainty. Aleatoric uncertainty remains captured

within each individual model through its likelihood formulation, while epistemic uncertainty is quantified by the disagreement across ensemble members [28]. This principle underlies a family of approaches that we refer to as *ensemble-style Bayesian approximations*.

Ensemble-style methods are attractive for two reasons. First, they are fully compatible with standard deep learning frameworks, requiring little to no modification of the training pipeline. Second, they substantially reduce the computational overhead compared to MCMC or SVI, while often providing competitive uncertainty estimates. They therefore represent a pragmatic approach inspired by theoretically rigorous but practically intractable Bayesian inference for large-scale neural architectures.

In this chapter, we focus on three representative methods: Monte Carlo Dropout (MCDO), Deep Ensembles (DEs), and Repulsive Last-Layer Ensembles (RLLEs). Monte Carlo Dropout and Deep Ensembles are well-established in the literature and serve as state-of-the-art reference points for scalable uncertainty estimation. By contrast, RLLEs are a more recent development [13], to which we have contributed, and we therefore provide a more detailed account of its formulation while referring to the original publication for further technical details. Together, these methods illustrate how ensemble-style approximations can address the computational bottlenecks of Bayesian neural networks.

9.1 Ensemble Methods

9.1.1 Monte Carlo Dropout (MCDO)

Among the earliest and most widely used ensemble-style approximations is Monte Carlo Dropout [175], which has become a standard baseline in uncertainty-aware deep learning due to its simplicity and minimal additional training overhead. It builds on the well-known regularization technique of dropout, where units are randomly set to zero during training to prevent overfitting, and extends it by applying dropout also at inference time. Gal and Ghahramani [174], [175] showed that dropout can also be interpreted as approximate variational inference, providing a Bayesian view of neural networks without modifying the underlying training procedure. Equivalently, each stochastic forward pass can be seen as evaluating one member of an implicit ensemble of subnetworks, where randomness

in the dropout masks induces predictor diversity [28]. By performing multiple such passes, one obtains a set of predictions that approximate samples from the posterior predictive distribution.

A key strength of MCDO is its practicality. It requires no changes to the training pipeline and incurs only minimal additional cost compared to a deterministic neural network. The method can even be retrofitted to pretrained models simply by enabling dropout at inference. Uncertainty estimates derived from the variability of the stochastic predictions have proven useful in applications such as out-of-distribution detection and active learning [147].

Despite these advantages, MCDO has important limitations. The variational family defined by dropout masks is relatively crude, which often leads to an underestimation of epistemic uncertainty and a weaker separation between epistemic and aleatoric components compared to more principled approaches such as SVI or MCMC. Its uncertainty estimates are further sensitive to the chosen dropout rate and network architecture. Moreover, while training remains inexpensive, inference typically requires a large number of stochastic forward passes to obtain sufficiently diverse samples. In practice, this number often exceeds that required by more sophisticated inference schemes, which, although more costly during training, represent the posterior diversity more efficiently and achieve comparable uncertainty quality with fewer samples.

In summary, MCDO provides a practical and lightweight approach to probabilistic deep learning, but its posterior approximation remains coarse, and its inference-time overhead can still be significant.

9.1.2 Deep Ensembles (DEs)

Deep Ensembles [154] represent another canonical baseline, widely regarded as the empirical state of the art for uncertainty estimation in deep learning, owing to their strong predictive performance and ease of use. The approach is conceptually simple: multiple networks are trained independently with different random initializations, and their predictions are aggregated to approximate Bayesian model averaging. Lakshminarayanan, Pritzel, and Blundell [154] formalized this idea under the name *Deep Ensembles*, showing that independently trained models with different random initializations can approximate sampling from diverse regions of the weight space. Aggregating their predictions yields an

implicit approximation to Bayesian model averaging.

Given an ensemble of M networks with parameters $\{\theta_m\}_{m=1}^{M}$, the predictive distribution for a new input x is estimated as

$$p(y \mid x, D) \approx \frac{1}{M} \sum_{m=1}^{M} p(y \mid x, \theta_m). \tag{9.1}$$

The resulting set of ensemble predictions constitutes approximate posterior samples, which can be analyzed with the same information-theoretic metrics introduced earlier to quantify total, epistemic, and aleatoric uncertainty.

Deep Ensembles have several appealing properties. They can be trained with standard pipelines and do not require specialized loss functions or hyperparameter tuning beyond what is used for a single deterministic network. In practice, they often achieve strong accuracy and well-calibrated uncertainty estimates across a wide range of tasks [88], [96], [154].

These benefits come at the cost of increased computation and storage. Training scales linearly with the number of ensemble members M, making ensembles M times more expensive than training a single deterministic model or using MCDO. Nevertheless, this is typically still cheaper than MCMC or SVI, where gradient-based sampling or optimization must be repeated for many posterior draws. At inference, ensembles require multiple forward passes, placing them on par with other approximate Bayesian methods in predictive cost. For prediction, ensembles also require storing M complete sets of network parameters, which scales memory linearly with ensemble size. This overhead is similar to MCMC, where multiple parameter samples from the chain must be maintained, but contrasts with MCDO and Gaussian SVI, which only need a single set of parameters (plus variance terms in the case of SVI).

While ensembles lack theoretical guarantees on covering the posterior, particularly in the high-dimensional parameter spaces of BNNs, their uninformed diversity often works surprisingly well in practice. Deep Ensembles are widely regarded as the empirical state of the art for large models where MCMC and SVI are not feasible. In fact, they can outperform variational methods when simple approximations such as mean-field Gaussians fail to capture multimodal posteriors, whereas independently trained ensemble members may collectively represent multiple modes [42], [77].

Overall, Deep Ensembles provide robust uncertainty estimates and remain

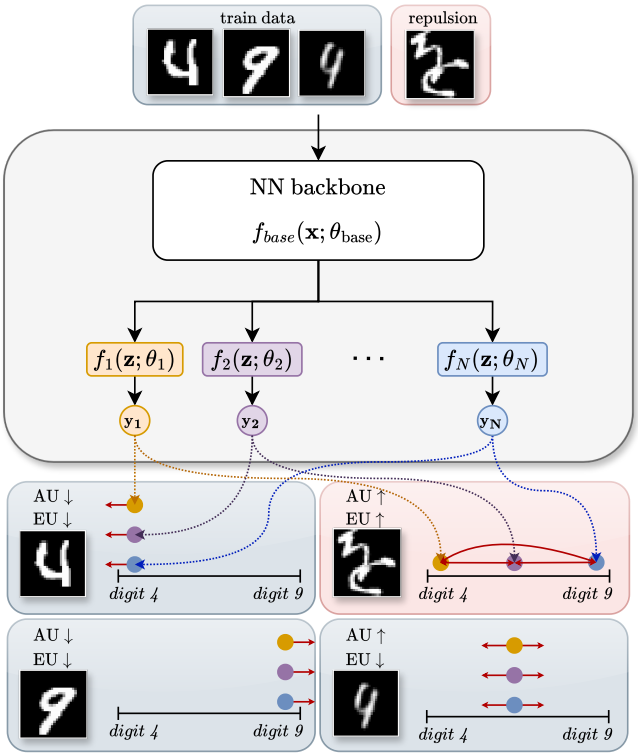


Figure 9.1 Illustration of a Repulsive Last-Layer Ensemble with N particles, where each output head corresponds to one particle (colored dots). Repulsion is enforced in function space using unlabeled samples from a different distribution, serving as *repulsion samples*. epistemic uncertainty (EU) is low when particles agree and increases with their spread, whereas aleatoric uncertainty (AU) arises from inherent label ambiguity, leading to predictions centered in uncertain probability regions. Reproduced with permission from [13].

the empirical state of the art in large-scale settings, albeit at significant training and storage cost. This motivates lighter alternatives that can retain ensemble diversity at a fraction of the overhead, such as Repulsive Last-Layer Ensembles, which we discuss next.

9.1.3 Repulsive Last-Layer Ensembles (RLLEs)

From Deep Ensembles to Repulsive Deep Ensembles. While Deep Ensembles provide strong empirical uncertainty estimates, their diversity is incidental, arising only from random initialization, stochastic optimization, or random shuffling of the training inputs. D'Angelo and Fortuin formalized this diversity by introducing *Repulsive Deep Ensembles* (RDEs) [37], which explicitly encourage ensemble members to remain diverse during training. Their method

augments the training loss with a repulsion term between predictive functions, grounded in Particle-Optimization Variational Inference (POVI) [93], [105], [114], [158], [182]. This ensures that ensemble members (the particles) do not collapse onto similar solutions, but instead spread out to approximate multiple modes of the posterior.

The key insight of RDEs is that *diversity in weight space is not sufficient*, since different parameter configurations can realize nearly identical functions. RDEs therefore emphasize *function-space diversity*, applying the repulsion term directly on the predictive distributions of each network. Empirical results demonstrated that such functional repulsion improves the estimation of epistemic uncertainty and out-of-distribution detection compared to standard DEs [37]. However, RDEs inherit the computational burden of classical ensembles: each network must still be trained and stored independently, making them costly for large-scale or resource-constrained applications.

Repulsive Last-Layer Ensembles. To address these computational limitations, Steger et al. proposed *Repulsive Last-Layer Ensembles* (RLLEs) [13]. Building directly on the principle of function-space repulsion from RDEs, RLLEs restrict diversity to the final prediction layer. Instead of training multiple independent networks, a single shared backbone is equipped with several output heads, each representing one ensemble member. Function-space repulsion is then applied only to these heads, ensuring predictive diversity at minimal additional cost. This architectural simplification drastically reduces training and storage overhead and integrates seamlessly with pretrained backbones, while retaining the theoretical motivation of RDEs.

Architecture and objective. Let $f_{\text{base}}(x;\theta_{\text{base}})$ denote a shared feature extractor and $\{f^{(i)}_{\text{head}}(\cdot;\theta^{(i)}_{\text{head}})\}_{i=1}^{n}$ denote n last-layer heads. Each particle function is $f^{(i)}(x) = f^{(i)}_{\text{head}}(f_{\text{base}}(x))$. Training minimizes the standard predictive loss plus a repulsive term that penalizes functional similarity across heads on a set of *repulsion samples*, thereby approximating function-space POVI with attraction–repulsion dynamics [13], [37]. This multi-head design avoids duplicating the backbone and reduces both training and memory cost compared to DEs [96], [154]. Figure 9.1 illustrates the concept.

Attraction–repulsion field in function space. Following POVI, the update field for the i-th particle (head) acts on the last-layer parameters $\theta_l^{(i)}$ and combines a data-driven attraction term with a kernel-based repulsion in function space [13], [37], [182]. We write

$$v\Big(\theta_l^{(i)}\Big) = \underbrace{\nabla_{\theta_l^{(i)}} \log p\Big(\theta_l^{(i)} \mid \mathcal{D}\Big)}_{\text{attraction}} - \gamma_{\text{repulsion}} \underbrace{\frac{\sum_{j=1}^{n} \nabla_{f\left(\theta_l^{(i)}\right)} k\Big(f\Big(\theta_l^{(i)}\Big), f\Big(\theta_l^{(j)}\Big)\Big)}{\sum_{j=1}^{n} k\Big(f\Big(\theta_l^{(i)}\Big), f\Big(\theta_l^{(j)}\Big)\Big)}}_{\text{repulsion in function space}}, \tag{9.2}$$

where $f(\theta_l^{(i)})$ denotes the predictive function of head i evaluated on a batch of repulsion samples, $\gamma_{\text{repulsion}}$ is the controllable repulsion strength, and k is a positive-definite kernel that measures functional similarity. A common choice is the RBF kernel applied to vector predictions on repulsion samples χ,

$$k\Big(f^{(i)}(\chi), f^{(j)}(\chi)\Big) = \exp\left(\frac{-\left\|f^{(i)}(\chi) - f^{(j)}(\chi)\right\|_p}{\nu}\right), \tag{9.3}$$

which yields a normalized repulsion term and directly encourages predictive diversity at the heads, while the backbone remains shared [13].

Choice of repulsion samples. A critical design choice is where to enforce functional repulsion. Applying it directly on the training data artificially inflates epistemic uncertainty in regions where the model should be confident and thus harms accuracy. Instead, RLLEs evaluate the repulsion term on unlabeled OOD data or on label-destroying augmentations, which probe regions that genuinely reveal epistemic uncertainty and thereby improve calibration and OOD detection. In high-dimensional settings, such auxiliary OOD or augmented samples serve as an effective approximation to full function-space coverage [13], [105].

Compatibility with pretrained backbones. Because diversity is confined to the last layer, RLLEs can be naturally applied to pretrained networks: freeze (or lightly fine-tune) the backbone, replace the classifier with n heads, and train with the function-space repulsion objective. This decouples representation learning from uncertainty-aware fine-tuning and is particularly effective when the

backbone is regularized to avoid feature collapse, e.g., via spectral normalization or related constraints [27], [128].

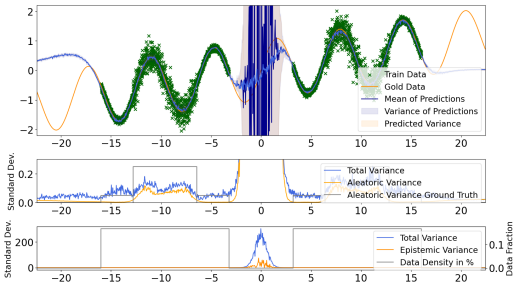
Context within related work. Repulsive ensembles can be instantiated in different spaces—weights, features, input gradients, or functions. RLLEs adopt the function-space variant while avoiding the cost of fully independent networks [37]. Compared to last-layer Bayesian baselines such as LL-Laplace or partially stochastic last layers, RLLEs employ explicit functional repulsion among multiple heads rather than a single probabilistic classifier [13], [38].

Limitations. Despite their advantages, RLLEs are not without caveats. Performance hinges on the calibration of repulsion strength and the quality and coverage of repulsion samples; poor choices can degrade accuracy or fail to elicit meaningful diversity [13]. Furthermore, because diversity is restricted to the last layer, expressiveness is more constrained than in fully independent DEs, especially when the shared backbone provides limited feature diversity. Nevertheless, the combination of function-space grounding, empirical competitiveness, and low computational footprint makes RLLEs compelling for scalable and embedded uncertainty estimation [13].

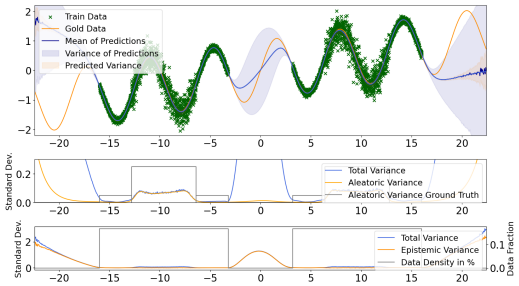
9.2 Comparative Evaluation

We first analyze the behavior of MCDO, DEs, and RLLEs on the Noisy Sine regression task, as studied in Simonides' master's thesis [6]. This controlled benchmark is particularly suited to assess the decomposition of epistemic and aleatoric uncertainty, since the ground-truth aleatoric variance is known and out-of-distribution regions are clearly identifiable.

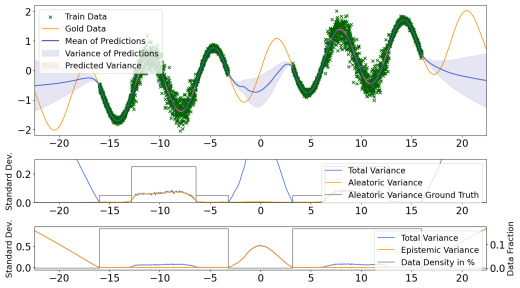
Overall, all three methods exhibit a marked sensitivity to the choice of activation function. MCDO performs worst: it fails to raise epistemic uncertainty in out-of-distribution regions, except for a single peak with the `SiLU` activation, and even then only in the central region rather than across the full support. DEs are considerably more robust, working reliably with several activations, and require only modest ensemble sizes ($M \approx 10$) to stabilize their predictions. RLLEs outperform MCDO but are more fragile than DEs: they require selecting a suitable activation (`SeLU` being most effective) and carefully calibrating their two



(a) MCDO



(b) Deep Ensembles



(c) RLLEs

Figure 9.2 Comparison on the Noisy Sine regression task with each method configured to its best-performing activation function and hyperparameters. MCDO: SiLU activation, dropout rate 0.15; DEs: M=10 members, SiLU activation; RLLEs: SELU activation with $\lambda_{\text{var}} = 0.25$ and $\gamma_{\text{repulsion}} = 1000$. MCDO systematically underestimates epistemic uncertainty, DEs provide stable and well-calibrated estimates with moderate ensemble size, and RLLEs achieve competitive performance when both activation and repulsion/variance parameters are carefully calibrated. Adjusted from [6].

hyperparameters, variance regularization and repulsion strength. This flexibility allows balancing aleatoric and epistemic contributions, but it also demands more tuning effort.

RLLEs demonstrated competitive overall performance, though their calibration proved more sensitive to hyperparameter settings than for Deep Ensembles. They also required substantially less training and storage cost than Deep Ensembles, while being only marginally more expensive than MCDO. Figure 9.2 illustrates these qualitative differences, showing the characteristic uncertainty profiles of all three methods in their best-performing configurations.

While the Noisy Sine task highlights qualitative differences in uncertainty behavior under controlled conditions, Dirty-MNIST provides a more realistic image-classification benchmark to illustrate these trade-offs at scale.

For illustration we reproduce results on the Dirty-MNIST benchmark (Ta-

Table 9.1 Comparison of uncertainty decomposition on Dirty-MNIST. Aleatoric and epistemic uncertainty are evaluated for detecting ambiguous and OOD samples. Best results are in bold, second best are underlined. RLLEs variants outperform Deep Ensembles (DE-5) clearly in OOD detection. Reproduced with permission from [13].

Method	Acc. ↑ [%]	NLL ↓ [%]	ECE ↓ [%]	OOD Auroc ↑ [%]		
				MNIST vs ambig. (AU)	MNIST vs. OOD (EU)	ambig. vs OOD (EU)
MAP	$79.97_{\pm 0.77}$	$58.12_{\pm 1.77}$	$2.82_{\pm 0.65}$	$93.83_{\pm 0.7}$	$97.71_{\pm 0.65}$	$76.11_{\pm 4.75}$
DDU	$79.97_{\pm 0.77}$	$58.12_{\pm 1.77}$	$2.82_{\pm 0.65}$	$93.83_{\pm 0.7}$	$\mathbf{99.78_{\pm 0.02}}$	$\mathbf{99.96_{\pm 0.01}}$
SNGP	$\underline{83.49_{\pm 0.11}}$	$49.78_{\pm 0.13}$	$3.98_{\pm 0.09}$	$88.76_{\pm 0.39}$	$94.68_{\pm 1.68}$	$73.88_{\pm 4.49}$
LL-Laplace	$80.73_{\pm 1.30}$	$55.90_{\pm 2.78}$	$2.03_{\pm 0.57}$	$94.3_{\pm 1.6}$	$98.41_{\pm 0.46}$	$93.15_{\pm 2.64}$
LL-POVI *(ours)*	$\mathbf{83.53_{\pm 0.16}}$	$\mathbf{48.32_{\pm 0.24}}$	$\mathbf{1.00_{\pm 0.14}}$	$\mathbf{96.82_{\pm 0.34}}$	$99.41_{\pm 0.22}$	$96.16_{\pm 1.53}$
RLL-POVI *(ours)*	$\mathbf{83.53_{\pm 0.16}}$	$\mathbf{48.32_{\pm 0.24}}$	$\mathbf{1.00_{\pm 0.14}}$	$\mathbf{96.82_{\pm 0.34}}$	$99.41_{\pm 0.22}$	$96.16_{\pm 1.53}$
fLL-POVI *(ours)*						
+ dirtyMNIST	$83.24_{\pm 0.20}$	$49.21_{\pm 0.29}$	$1.18_{\pm 0.12}$	$96.38_{\pm 0.34}$	$99.29_{\pm 0.43}$	$95.36_{\pm 2.82}$
+ eMNIST	$\underline{83.52_{\pm 0.20}}$	$48.91_{\pm 0.24}$	$1.18_{\pm 0.20}$	$95.27_{\pm 2.07}$	$99.3_{\pm 0.3}$	$\underline{99.52_{\pm 0.23}}$
+ Patches-16	$83.51_{\pm 0.16}$	$\underline{48.35_{\pm 0.24}}$	$\underline{1.03_{\pm 0.13}}$	$\underline{96.74_{\pm 2.14}}$	$\underline{99.52_{\pm 0.26}}$	$97.69_{\pm 1.38}$
+ Patches-8	$\underline{83.52_{\pm 0.15}}$	$48.40_{\pm 0.24}$	$\underline{1.02_{\pm 0.15}}$	$96.63_{\pm 1.81}$	$99.4_{\pm 0.25}$	$98.59_{\pm 0.61}$
+ Patches-4	$\underline{83.50_{\pm 0.18}}$	$48.59_{\pm 0.23}$	$\underline{1.05_{\pm 0.17}}$	$96.44_{\pm 2.0}$	$99.45_{\pm 0.21}$	$99.09_{\pm 0.38}$
DE-5	$83.31_{\pm 0.15}$	$50.26_{\pm 0.40}$	$5.01_{\pm 0.67}$	$96.23_{\pm 0.15}$	$98.96_{\pm 0.2}$	$93.88_{\pm 1.57}$

ble 9.1), whereas the original publication [13] reports a comprehensive evaluation across multiple datasets and baselines.

Across the broader set of experiments, the repulsive variant (RLL-POVI) emerged as the strongest among last-layer POVI methods, consistently outperforming plain and functional variants. Performance was particularly boosted by using auxiliary repulsion samples such as image patches, underlining that repulsion sample choice is critical for eliciting meaningful epistemic diversity. Overall, RLLEs matched or exceeded the uncertainty quality of DEs at a fraction of the computational and storage cost, and their compatibility with pretrained backbones makes them attractive for practical deployment.

9.3 Hardware Evaluation with TVM

The final step in our evaluation is to assess how well Repulsive Last-Layer Ensembles can be deployed on embedded hardware and what performance they achieve in practice. Because uncertainty-aware methods are only useful in constrained environments if they remain computationally efficient, we benchmark RLLEs on ARM Cortex-A series processors and compare them against deterministic baselines and Deep Ensembles.

One advantage of RLLEs is their compatibility with existing compiler toolchains. Unlike the Probabilistic Forward Pass, which required custom operators to be

Table 9.2 Benchmark RLLEs vs. DEs performance on embedded ARM CPUs.

Method	Dataset	Arch.	n	CPU	MACs [M]	Latency [ms]	
						naive	tuned
Det. NN	Noisy Sine	MLP	-	A72	2.6	9.14	0.99
RLLE	Noisy Sine	MLP	10	A72	7.8	25.93	2.93
DE-10	Noisy Sine	MLP	10	A72	26.0	91.43	9.92
Det. NN	Noisy Sine	MLP	-	A76	2.6	4.05	0.46
RLLE	Noisy Sine	MLP	10	A76	7.8	11.31	1.22
DE-10	Noisy Sine	MLP	10	A76	26.0	40.53	4.58
Det. NN	Dirty-MNIST	LeNet-5	-	A72	36.0	59.43	8.17
RLLE	Dirty-MNIST	LeNet-5	10	A72	37.0	62.41	7.94
DE-10	Dirty-MNIST	LeNet-5	10	A72	360.5	594.31	81.69
Det. NN	Dirty-MNIST	LeNet-5	-	A76	36.0	26.50	2.55
RLLE	Dirty-MNIST	LeNet-5	10	A76	37.0	27.88	2.96
DE-10	Dirty-MNIST	LeNet-5	10	A76	360.5	265.04	25.49

integrated into the TVM stack, RLLEs rely exclusively on standard deep learning components. This makes the compilation process straightforward: models can be imported into TVM without modification, and performance tuning can be carried out using standard auto-scheduling methods. In our experiments we employed the *Meta Scheduler* [32] from the TVM compiler stack [26], [79], [115] to automatically optimize operator schedules for the specific ARM architectures, which proved essential for achieving efficient execution. Untuned schedules left substantial performance untapped, whereas auto-scheduling consistently reduced latency and produced efficient execution across all tested configurations.

Table 9.2 summarizes MAC counts and inference latencies for both tasks. For the Noisy Sine task, evaluated with a mini-batch size of 1000, we used a two-layer MLP with 50 hidden neurons per layer, while the more demanding Dirty-MNIST task, evaluated with a mini-batch size of 128, was based on LeNet-5. The complexity of the backbone architecture determines the relative overhead of the ensemble heads. In the small MLP, ten heads dominate the network, increasing the total size by roughly a factor of three in terms of MACs. In contrast, for LeNet-5 the additional heads introduce only a 2.7% overhead. This highlights a central advantage of RLLEs: their cost does not scale with the backbone architecture, making them suitable even for very large models.

Repulsive Last-Layer Ensembles reduce training and storage requirements compared to Deep Ensembles, while retaining competitive predictive performance. On hardware, they achieved 3.4× faster inference on the Noisy Sine task and 8.1×

Table 9.3 Performance PFP vs. RLLEs.

Dataset	Arch.	CPU	Batch Size	Latency [ms]	
				PFP	RLLE
Dirty-MNIST	LeNet-5	A72	1	1.23	0.51
			8	7.77	0.91
			16	15.58	1.32
			32	32.91	2.71
			64	70.82	4.89
			128	150.72	8.00
		A76	1	0.50	0.27
			8	3.03	0.41
			16	6.33	0.46
			32	13.60	0.63
			64	27.46	1.11
			128	61.92	2.93

faster on Dirty-MNIST. Compiler-level optimizations such as Meta Scheduler tuning further improved latency, but the core efficiency arises from the method itself: by sharing a backbone and introducing diversity only in the last layer, RLLEs provide uncertainty estimation at a fraction of the computational cost. This makes them a practical and scalable choice for deploying uncertainty-aware models on embedded systems.

9.3.1 Comparison of PFP and RLLEs

Finally, we compare PFP and RLLEs, the two BNN approximation methods examined in detail throughout this work. Both were benchmarked on the Dirty-MNIST dataset with a LeNet-5 backbone to directly assess uncertainty quality and computational performance.

In terms of uncertainty estimation, both methods perform strongly. For out-of-distribution detection, PFP reached an AUROC of 0.966, while RLLEs achieved 0.995, highlighting that both approximate Bayesian inference and ensemble-based approaches yield consistent uncertainty calibration across in- and out-of-distribution regimes.

The computational comparison, summarized in Table 9.3, highlights the distinct performance characteristics of the two approaches. All reported measurements were obtained from TVM builds tuned with the Meta Scheduler [32], which substantially reduced latency compared to untuned execution for both methods. RLLEs consistently outperform PFP, and the performance gap widens

with increasing batch size. For small mini-batches, both methods achieve low latency, with RLLEs already showing a modest advantage that becomes increasingly pronounced as batch size grows. This difference arises from the underlying computational structure of the two methods. RLLEs rely on standard dense operations, dominated by matrix multiplications that are natively supported by highly optimized hardware kernels and thus scale effectively with batch size through vectorization and parallelization. Their multi-head design further exposes an additional degree of parallelism: the output heads can be processed largely independently on separate processor cores, as they share the same backbone features but have distinct final layers. This enables partial parallel execution without inter-head communication, improving utilization on multi-core embedded platforms while maintaining a small memory footprint. In contrast, PFP propagates both means and variances through each layer, which increases the number of intermediate tensors and memory accesses. While parts of the computation can be expressed as matrix multiplications, the frequent reading and writing of mean–variance pairs limits cache locality. Moreover, the inherently sequential propagation of stochastic moments offers fewer opportunities for parallel execution than the independent output heads of RLLEs. As a result, PFP achieves lower throughput at larger batch sizes, despite its efficient formulation at the algorithmic level.

Conceptually, the two methods occupy complementary positions in the landscape of efficient Bayesian inference. PFP remains within the variational framework and offers a principled, calibration-stable approximation that eliminates stochastic sampling entirely through closed-form propagation. RLLEs, on the other hand, embrace an ensemble-based view, retaining the multi-modality of Deep Ensembles while remaining compatible with standard toolchains and hardware accelerators. This design yields remarkable speed and flexibility, though at the expense of increased sensitivity to calibration choices such as the repulsion strength and the selection of repulsive samples.

Taken together, the results underline that neither approach dominates universally. The Probabilistic Forward Pass offers theoretical rigor and intrinsic stability, whereas RLLEs achieve greater empirical efficiency and scalability. Both methods therefore represent viable and complementary pathways for enabling uncertainty-aware inference on embedded systems.

Summary

This chapter has examined ensemble-based approximations for Bayesian neural networks, focusing on MCDO, DEs, and RLLEs. Our controlled experiments on the Noisy Sine task confirmed the known weaknesses of MCDO, the robustness and empirical strength of DEs, and the potential of RLLEs to achieve competitive uncertainty estimates at a fraction of the training and storage cost. While RLLEs demand careful calibration of activation functions, hyperparameters and well chosen repulsive samples, their design provides explicit flexibility to balance epistemic and aleatoric uncertainty.

On the hardware side, we showed that RLLEs integrate seamlessly with standard compiler toolchains. Thanks to their reliance on standard operators, they can be compiled and tuned with TVM directly, turning their theoretical efficiency into practical speedups on embedded CPUs. Because the additional heads add only marginal overhead compared to larger backbones, RLLEs scale favorably to larger architectures and remain well suited for deployment under tight resource constraints.

In summary, RLLEs strike a pragmatic balance between theoretical grounding and practical deployability. They retain much of the uncertainty quality of full ensembles while reducing computational cost to levels compatible with embedded devices.

The following chapter extends this perspective on probabilistic inference beyond digital accelerators, investigating photonic hardware where inherent device noise is harnessed directly as a stochastic source for probabilistic inference.

10

Probabilistic Photonic Computing for Bayesian Neural Networks

Information is not a disembodied abstract entity;
it is always tied to a physical representation.

— Rolf Landauer, *The Physical Nature of Information* (1996)

In the preceding chapters we investigated two complementary strategies to reduce the cost of deploying neural networks on resource-constrained devices. From the algorithmic side, we developed methods to reduce the number of costly computations, while from the hardware side, we considered approaches that lower the cost of the operations themselves. For deterministic deep neural networks, this two-pronged perspective was reflected by *Galen* as an automatic compression method on the algorithmic level [21], and by analog accelerators as a means to perform the underlying matrix multiplications more efficiently [4], [7]–[9], [44].

In this chapter we extend this perspective to Bayesian neural networks. Algorithmic approximations such as partial BNNs [24], [65], RLLEs [13] and the PFP [3], [186] reduce the computational burden of probabilistic inference. At the

hardware level, analog computing platforms promise to accelerate BNN inference in a complementary way, by embedding probabilistic operations directly into physical processes.

This perspective is particularly appealing because BNNs require stochasticity at their core. On digital processors, randomness must be simulated by pseudo-random number generators and integrated into multiple forward passes, which adds substantial computational overhead. Analog hardware, by contrast, is inherently noisy. While this noise is typically seen as a liability for deterministic inference, for probabilistic inference it can serve as a natural entropy source. If the noise can be shaped and controlled, the probabilistic sampling step of BNNs can be executed directly in hardware, making uncertainty estimation both faster and more energy efficient.

Photonic computing offers a compelling platform for this idea. Light not only enables extremely high bandwidth and parallelism—for example through wavelength-division multiplexing—but also provides accessible entropy sources in the form of intensity fluctuations, phase noise, or quantum effects [1]. These processes can be leveraged to realize the stochastic sampling required by Bayesian inference, aligning the algorithmic demands of probabilistic models with the physical properties of the hardware.

This chapter builds on two recent joint works that we conducted in collaboration with researchers specializing in photonic hardware. The position paper *Probabilistic Photonic Computing for AI* [1] and the proof-of-concept demonstration *Probabilistic Photonic Computing with Chaotic Light* [10] emerged from this collaboration. The projects were carried out in a co-design effort, combining our expertise in probabilistic modeling and Bayesian neural networks with our partners' expertise in photonic device design and experimental realization. Accordingly, this chapter emphasizes the proof of concept as a hardware-accelerated BNN inference engine and highlights the algorithmic adaptations required to map Bayesian neural networks onto photonic hardware. For detailed discussions of the photonic design choices and device physics, we refer the reader to the original publications.

10.1 Photonic Neural Network Inference

BNNs provide a principled framework for predictive uncertainty but incur high inference cost due to repeated sampling and multiple forward passes. On digital processors, pseudo-random number generation and sample-wise evaluation add latency and energy overhead. Algorithmic approaches reduce or sidestep repeated sampling, thereby lowering the computational burden, yet they rarely eliminate the fundamental need for stochasticity in probabilistic inference.

Analog computing platforms, including electrical accelerators [4], [44], memristive crossbars [84], [109], and photonic processors [124], [164], promise substantial energy efficiency compared to digital hardware. However, they are inherently subject to imperfections such as nonlinearities, device variations, and computational noise [7]–[9]. For deterministic inference, such noise degrades accuracy and requires countermeasures like noisy or hardware-in-the-loop training. For probabilistic inference, by contrast, noise is not necessarily a limitation: it can serve as an entropy source and thus as the very building block of Bayesian computation. This shift in perspective, from mitigating noise to exploiting it, motivates the exploration of analog hardware for accelerating probabilistic inference.

Photonic computing offers properties that are particularly attractive for this idea. Light propagates at extremely high speed and enables massively parallel signal processing through wavelength-division multiplexing (WDM). The available optical bandwidth in the telecom band alone exceeds several THz, far surpassing electronic bandwidths [1]. Moreover, photonic systems inherently exhibit entropy sources, including phase noise, intensity fluctuations in chaotic light, and quantum fluctuations at the single-photon level [229], [230], [247], [249]. These noise processes can be harnessed as true random number generators, avoiding the overhead of digital pseudo-random generators and directly linking stochasticity to the physical information carrier.

A wide range of photonic neural architectures has been proposed over the past decade. Coherent nanophotonic meshes based on Mach–Zehnder interferometers (MZIs) enable programmable optical linear algebra [164], while diffractive optical networks perform passive all-optical inference [124]. Microring weight banks and crossbar arrays based on phase-change materials (PCMs), in particular germanium–antimony–telluride (GST), allow compact in-memory optical multi-

plication [64], [129]. These works primarily target deterministic inference, but they establish the device toolkit that can also support probabilistic computing.

Entropy sources are a central ingredient of probabilistic computing. Classical statistical optics established the photon statistics of coherent and chaotic fields [230], [247], [249], while later studies investigated amplified spontaneous emission (ASE) statistics in dense wavelength-division multiplexing regimes relevant to telecom applications [229]. Building on these foundations, chaotic light has been identified as a particularly practical entropy source, since it can be generated in the telecom band by erbium-doped fibers or waveguides. Unlike digital pseudo-random generators, chaotic-light entropy is broadband, spans multiple THz, and is directly compatible with optical processors [10].

In parallel, analog non-volatile memories such as electronic memristors have been proposed as hardware substrates for probabilistic neural inference. Randomness in switching processes and conductance drift can be interpreted as tunable stochastic weights, and recent work has demonstrated Bayesian inference with memristor-based neural networks [15]. These works exploit stochasticity in electronic devices, reinforcing the broader trend of turning hardware noise from a liability into a computational resource.

Our recent works on *probabilistic photonic computing for AI* [1] and on a prototype implementation using *chaotic light* [10] extended this concept to photonic systems. We introduced chaotic light as a controllable entropy source and demonstrated its integration with GST-based photonic crossbars, thereby establishing photonic processors as a viable platform for probabilistic inference.

10.2 Hardware Design Principles

Photonic systems expose multiple noise sources that can be harnessed for probabilistic computing [1]. Among the most relevant are phase noise in coherent lasers, intensity noise in chaotic light sources, and quantum fluctuations at the single-photon level. All three provide access to physical entropy, but they differ in ease of integration with large-scale photonic processors.

Phase noise can be exploited through interferometric readout schemes and has been used for photonic random number generation. However, interferometers impose strict stability requirements, and integrating phase-sensitive components at scale remains challenging. Quantum fluctuations, such as those exploited in

true photon-number–resolving detectors, provide intrinsic randomness but require cryogenic operation and highly sensitive components, making them impractical for near-term integrated photonic processors. In contrast, chaotic light generated by amplified spontaneous emission (ASE) is broadband, robust, and easily available in the telecom band. Its intensity fluctuations follow Bose–Einstein statistics and can be directly measured with standard photodetectors. For these reasons, chaotic light was chosen as the entropy source in the prototype of [10].

To enable matrix multiplication, the processor employs a photonic crossbar array. Weights are stored in the non-volatile phase-change material germanium–antimony–telluride, which provides multiple stable transmission levels. Multiplication is realized by attenuating the optical carrier according to the programmed transmission state (Fig. 10.1a), and addition is performed by summing overlapping optical fields in a waveguide. A photograph of the fabricated chip is shown in Fig. 10.1b, while the full crossbar structure is depicted in Fig. 10.1c.

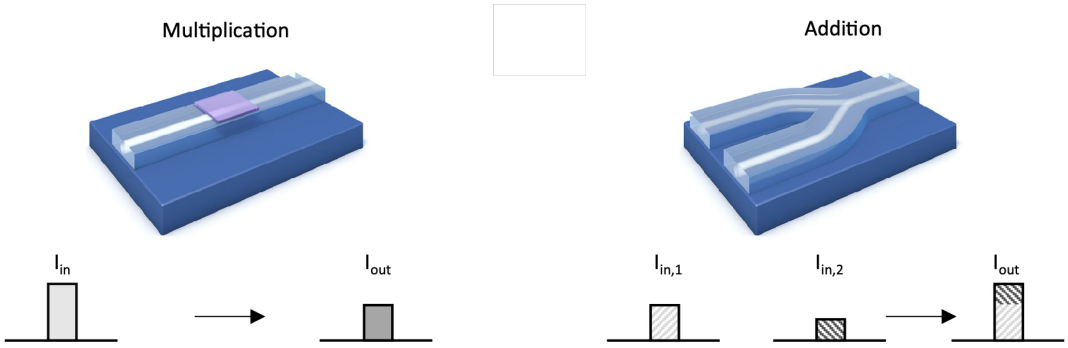
Parallelism is achieved by wavelength-division multiplexing. The broadband spectrum of chaotic light spans several THz, which can be demultiplexed into distinct wavelength channels. Each channel carries an independent realization of the chaotic fluctuations, enabling multiple uncorrelated random samples to be processed simultaneously. This property directly accelerates probabilistic inference, where many stochastic samples are typically required.

Finally, several practical constraints shape the design. The independence of random numbers is limited by the correlation time of chaotic fluctuations and the electrical bandwidth of the detectors. Residual correlations between wavelength channels can occur due to imperfect demultiplexing. Moreover, photonic components such as waveguides and detectors remain larger than electronic counterparts, imposing constraints on integration density [1].

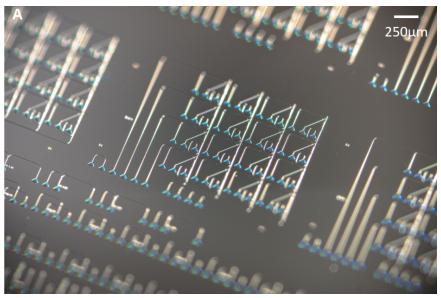
In summary, chaotic light was selected as a pragmatic entropy source due to its controllability and ease of integration with GST-based photonic crossbars. Combined with WDM for parallelization, these design choices establish a physical platform for accelerating probabilistic inference on photonic hardware (Fig. 10.1).

10.3 Making Noise Controllable

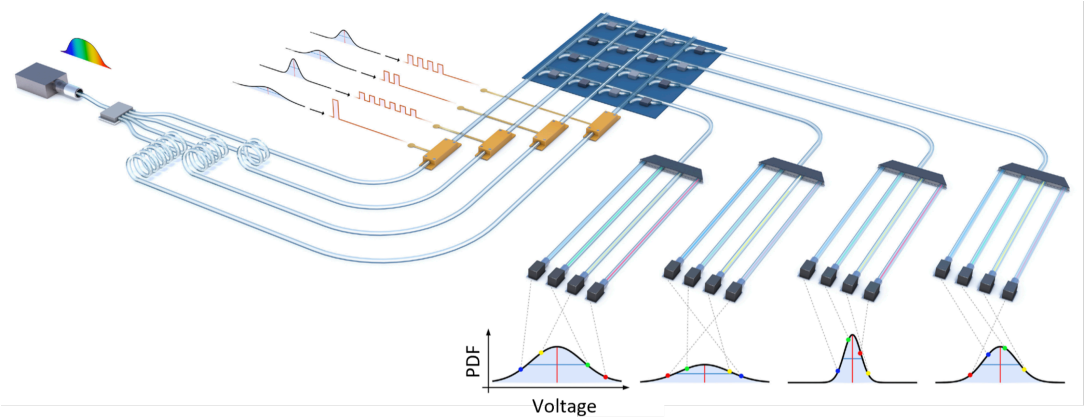
A central challenge in harnessing physical noise for probabilistic computing lies not only in accessing it but in making it controllable. The measured signal at



(a) Multiplication and Addition

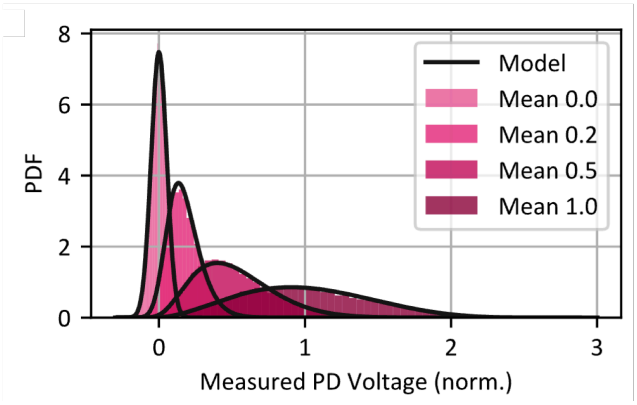


(b) Chip Photograph

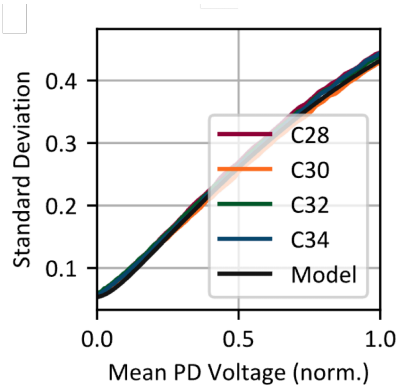


(c) Crossbar Structure

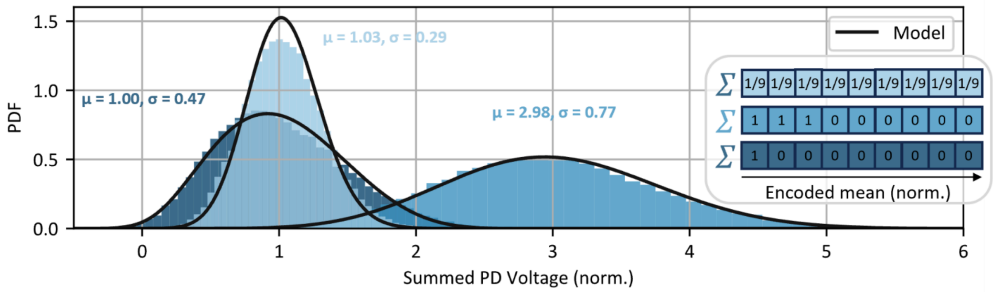
Figure 10.1 Photonic crossbar with GST weights. Subfigure a illustrates how multiplication is realized by attenuating the optical carrier according to the programmed GST state, while addition is implemented by overlapping multiple optical fields in a waveguide, consistent with the statistical behavior of chaotic light distributions reported in [10]. Subfigure b shows a microscope photograph of the fabricated photonic chip. Subfigure c provides a three-dimensional overview of the full crossbar structure, where chaotic light from an ASE source is split and delayed to form multiple uncorrelated optical carriers. Input distributions are encoded as waveforms on these carriers, processed through the GST-based crossbar, and demultiplexed by WDM into independent sampling channels. Reproduced with permission from [10].



(a) Measured PDFs at different mean intensities



(b) Variance–mean coupling



(c) Variance control by encoding scheme

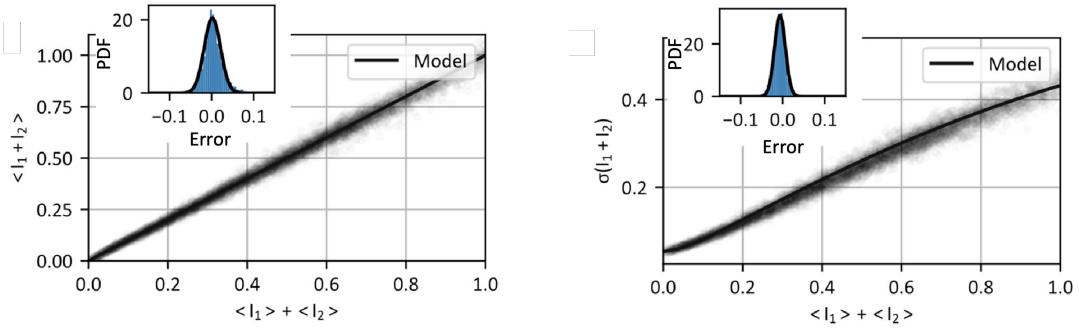
Figure 10.2 Statistical properties of chaotic-light noise. Subfigures a and b confirm Bose–Einstein statistics and the quadratic dependence of the variance on the mean intensity. Subfigure c illustrates controllable-noise encoding, where the same total intensity yields distributions with different variances. Reproduced with permission from [10].

the photodetector combines two contributions: electronic noise from the readout circuit and intensity fluctuations of the chaotic-light carrier. At small optical intensities, the measured distribution is dominated by electronic noise, which is approximately Gaussian and independent of the mean intensity. With increasing intensity, photonic fluctuations dominate and the distributions evolve towards the Bose–Einstein form expected from chaotic light [10], [230], [247], [249].

The beating of frequency components in amplified spontaneous emission leads to photon-number fluctuations described by an M-fold Bose–Einstein distribution [10]. Its variance grows quadratically with the mean intensity,

$$\mathrm{Var}(n) = n_{\mathrm{mean}} + \frac{n_{\mathrm{mean}}^2}{M}, \tag{10.1}$$

where n_{mean} is the average photon number detected within the measurement



(a) Mean (b) Standard Deviation

Figure 10.3 Experimental validation of addition with noisy signals. Subfigure a shows that the means add linearly, as expected. In contrast, Subfigure b shows that the standard deviations scale nonlinearly due to the constant electronic ground noise contribution. Reproduced with permission from [10].

interval and directly corresponds to the optical intensity, and M is the degeneracy factor, approximated by the ratio of the measurement time T to the coherence time τ_c of the source [10]. In the presented setup $M \approx 10.64$.

As a consequence of this quadratic relation, mean and variance cannot be tuned independently. This intrinsic coupling poses a challenge when mapping probabilistic operations onto hardware, since BNNs require distributions with independently adjustable parameters.

This behavior is confirmed experimentally (Fig. 10.2): at low means the distributions are narrow and Gaussian-like, while at higher means they broaden and follow Bose–Einstein statistics (Fig. 10.2a, Fig. 10.2b). Detector saturation further limits the width at large means (Fig. 10.2b). Another property becomes evident when input distributions are accumulated (Fig. 10.3). The means add linearly (Fig. 10.3a), whereas the standard deviations follow a nonlinear curve due to the mean-invariant electronic noise contribution (Fig. 10.3b) [10].

Building on this principle, we propose a scheme to control the variance independently of the mean. Where not only the number of active slots but also their amplitudes are adjusted. The same total intensity can either be represented by a few strong impulses, introducing proportionally more noise, or by many smaller impulses, where the noise averages out and becomes sub-proportional. As a result, the same intensity can yield output distributions with different variances (Fig. 10.2c). This controllable-noise encoding enables a decoupling of mean and variance, within the bounds set by the physical noise sources, and thus renders chaotic-light noise tunable.

10.4 Adapting BNNs to Photonic Hardware

The feasibility of photonic probabilistic inference depends not only on the availability of tunable noise but also on adapting Bayesian neural network operators to the constraints of the hardware. Rather than assuming a generic architecture, the Bayesian neural network must be co-designed with the photonic device to exploit its strengths while compensating for its limitations.

The first design choice is to realize stochasticity through probabilistic activations rather than probabilistic weights. Conventional BNNs often assign distributions to weights, but this strategy is infeasible in our setting for two reasons. The prototype crossbar provides only a limited number of programmable weight cells, and although phase-change material devices based on GST are well suited for compact in-memory multiplication, their transmittance can only be reprogrammed slowly compared to photonic processes. Rapidly varying weights for sampling is therefore not a viable option. In contrast, the chaotic-light carrier produces intrinsic intensity fluctuations that are directly detected at the photodiode readout. These fluctuations supply random samples at high rate, and their variance can be controlled by the proposed encoding scheme. It is thus more efficient to keep the weights deterministic and implement randomness at the activation level.

A second design choice concerns numerical precision. The optical crossbar supports only a limited number of transmission levels, and the electrical readout is quantized. To maintain robustness under these conditions, quantization-aware training (QAT) [120], [122] is required so that the BNN representation matches the effective bit-width of the hardware. Quantization for BNNs remains largely unexplored, but our evaluation shows that the behavior is similar to conventional deep neural networks: once the bit-width falls below a critical threshold, accuracy and uncertainty estimation collapse together. This threshold depends on both the architecture and the dataset. Nevertheless, provided the bit-width remains above this limit, QAT enables reliable inference under reduced precision.

A third design choice addresses the noise characteristics of the system. The chaotic-light fluctuations are tunable only within a fixed range, and on top of this comes a constant base noise floor mainly from the electronic components. As a result, the BNN can adjust activation variances only within these bounds, while the residual base noise must be learned to tolerate during hardware-model

training.

A fourth design choice arises from the strictly positive nature of the optical signals. In this setup, light can only be added but not subtracted, so the hardware enforces a positive-only domain. As a consequence, negative weights and activations are excluded and must be compensated for at the algorithmic level.

Finally, the limited chip area of the prototype constrains the number of weight cells. This rules out large-scale dense or convolutional layers. Instead, we developed a probabilistic average-pooling operator that requires only a few weights but exposes a large number of stochastic degrees of freedom by allowing noise parameters to be tuned on a per-activation level.

Figures 10.4 and 10.5 illustrate the probabilistic photonic average-pooling operator in its two complementary forms: a training model and a physical simulation model, both implemented in Pyro. The coexistence of both implementations reflects the need to reconcile efficient gradient-based optimization in software with a faithful representation of the stochastic behavior observed on photonic hardware.

The training model employs a Gaussian approximation of the Bose–Einstein distribution to avoid an additional sampling step and to accelerate training, while remaining sufficiently close to the physical statistics. This abstraction enables hardware-aware SVI-based training with realistic noise characteristics at a manageable computational cost. In contrast, the physical model samples directly from the exact Bose–Einstein PDF, includes stochastic quantization, and explicitly accounts for photonic imperfections and limited precision. Together, these two operator variants ensure consistency between training and hardware-level simulation, allowing the BNN to be optimized efficiently in software and evaluated accurately on the photonic prototype.

In summary, adapting Bayesian neural networks to photonic hardware requires several key design choices, as exemplified by our prototype implementation: stochasticity is implemented in the activations rather than the weights, representations are trained with reduced precision in mind, activation variances are adjusted only within constrained bounds, the positive-only nature of optical signals is considered, and operators are tailored to the limited number of available weights. Together, these adaptations enable a proof-of-concept demonstration of probabilistic inference on a photonic accelerator.

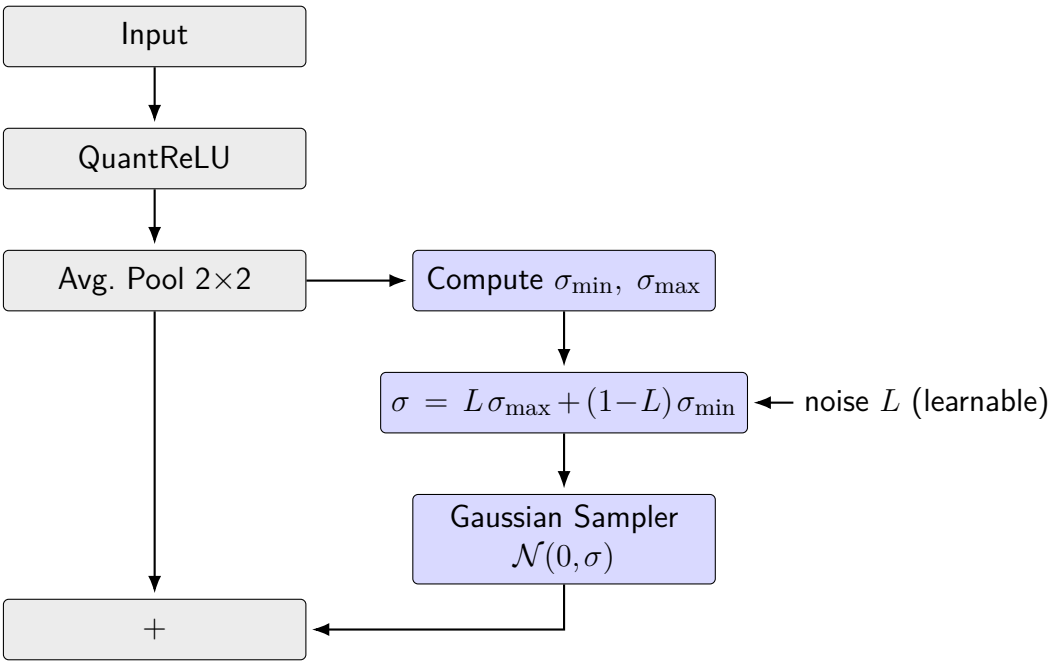


Figure 10.4 Photonic probabilistic average-pooling operator (*training model*). Deterministic blocks (gray) compute activations via QuantReLU and average pooling. This model is used during training with Pyro; the Gaussian approximation of the Bose–Einstein distribution avoids an additional sampling step and significantly accelerates training while preserving the learned distributional characteristics. Reproduced with permission from [10].

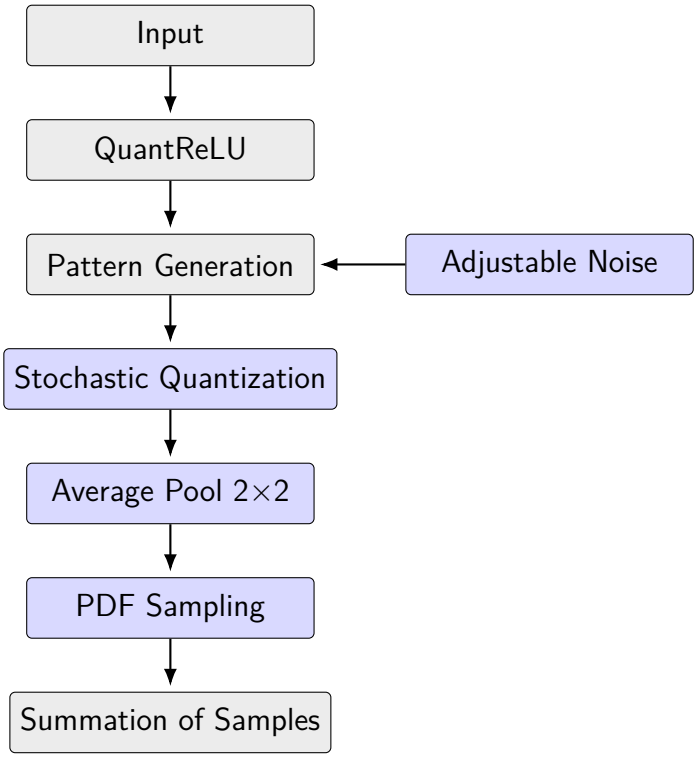


Figure 10.5 Photonic probabilistic average-pooling operator (*physical simulation model*). Deterministic preprocessing (gray) is followed by probabilistic operations (blue), where randomness is injected from the chaotic-light noise source. The layer performs stochastic quantization, average pooling, and PDF-based sampling using the exact Bose–Einstein distribution, with cached PDFs reused to reduce computational overhead. This version models the hardware-level behavior including photonic imperfections and quantization effects. Reproduced with permission from [10].

10.5 Experimental Demonstration

To demonstrate the feasibility of probabilistic inference on photonic hardware, we performed a simple OOD detection experiment on a modified MNIST dataset. Figure 10.6 illustrates the setup: digits 0–8 were used for training and in-distribution testing, while digit 9 was held out as the OOD class. This task highlights both classification accuracy and the ability of the network to signal uncertainty when facing unseen inputs.

The experiment was carried out with a compact BNN tailored to the photonic prototype. Figure 10.7 shows the network architecture, which is based on a modified LeNet-5. Deterministic layers (gray) are combined with probabilistic average-pooling layers (orange) as introduced in the previous section. This design

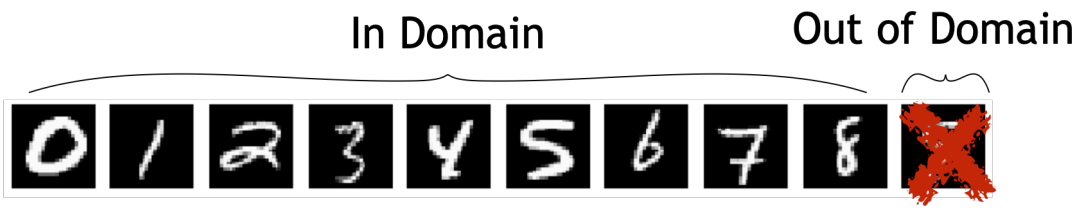


Figure 10.6 Illustration of the 9-class MNIST OOD detection task. Digits 0–8 are used for training and ID testing, while digit 9 is held out as the OOD class. Reproduced with permission from [10].

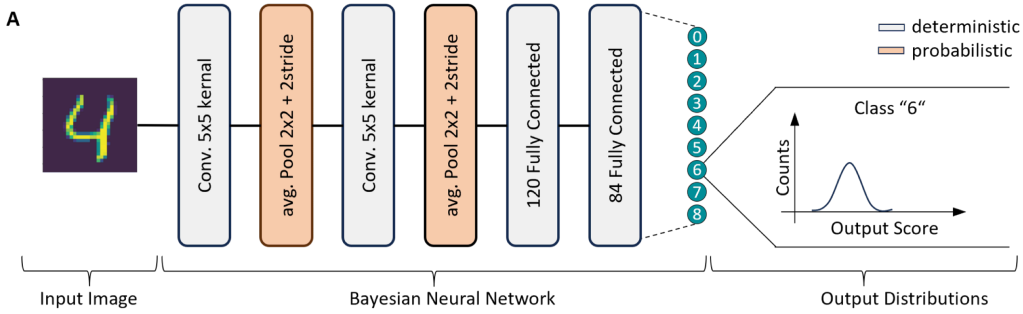


Figure 10.7 BNN architecture for the MNIST OOD experiment. Deterministic layers (gray) are combined with probabilistic average-pooling layers (orange), implemented using the probabilistic max-pool operator. Reproduced with permission from [10].

allows stochasticity to be injected into the activations through chaotic-light sampling while keeping the number of photonic weights small.

Training dynamics further illustrate how the network learns both classification and uncertainty separation. As shown in Fig. 10.8, accuracy on in-domain digits rises quickly, while the separation in mutual information between ID and OOD inputs develops more gradually. This growing gap in mutual information demonstrates that the photonic BNN learns not only to classify seen digits but also to identify unseen ones through uncertainty estimates.

To illustrate this effect qualitatively, Fig. 10.9 shows predictive distributions for a representative in-distribution digit (4) and an out-of-distribution digit (9). The in-distribution prediction is sharply peaked at the correct class, while the OOD prediction is broad and uncertain, consistent with the role of mutual information as an OOD indicator.

The separation becomes most apparent when analyzing the full test set. Figure 10.10 demonstrates that ID and OOD examples form two clearly separated peaks in terms of mutual information, enabling reliable OOD detection based on

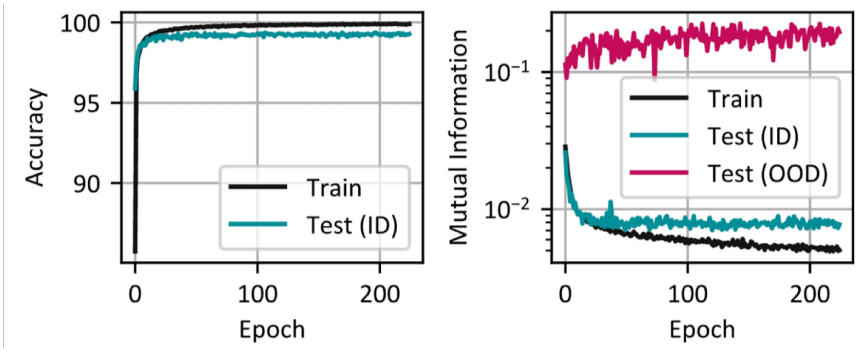


Figure 10.8 Training and test dynamics of the MNIST OOD experiment. Left: accuracy on training and ID test data. Right: separation between ID and OOD data in terms of mutual information. Reproduced with permission from [10].

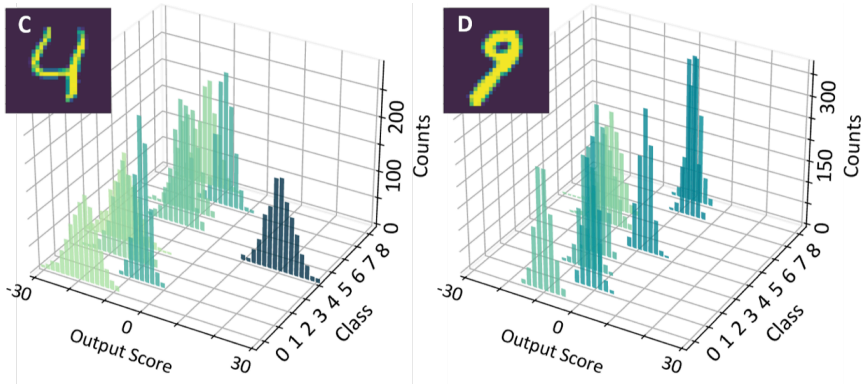


Figure 10.9 Predictive distributions for an ID (left) and an OOD (right) sample. In-domain predictions are sharply peaked, while OOD predictions are spread out, reflecting uncertainty. Reproduced with permission from [10].

uncertainty.

Together, these results confirm that probabilistic inference with photonic hardware is feasible. Even with the limitations of the prototype, the network achieves high accuracy on in-distribution data and successfully identifies out-of-distribution samples via uncertainty estimates. The experiment demonstrates that chaotic-light-based activations provide a reliable entropy source for Bayesian neural networks and that photonic accelerators can deliver uncertainty-aware inference.

Summary

Building on the algorithmic foundations of probabilistic inference and the modeling of analog noise in previous chapters, this chapter has explored the feasibility

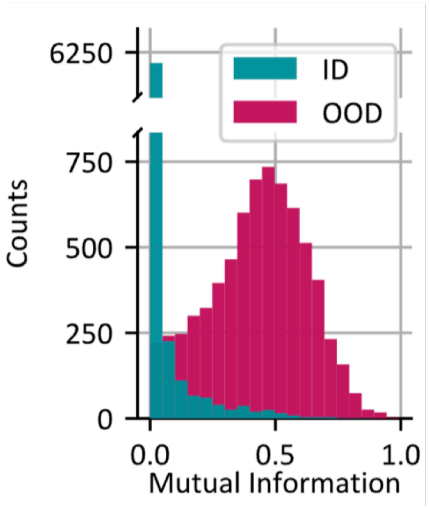


Figure 10.10 Mutual information histogram for ID and OOD test samples. The clear separation between the two distributions demonstrates the ability of the photonic BNN to detect unseen data. Reproduced with permission from [10].

of implementing Bayesian neural networks on photonic hardware. The motivation arises from reinterpreting the existing tension between the high computational cost of probabilistic computing in digital inference and the intrinsic, yet typically suppressed, noise present in analog accelerators.

The core idea developed here is to invert this perspective and exploit the intrinsic stochasticity of photonic systems as a computational resource. In particular, chaotic-light intensity fluctuations, when made controllable, provide a natural entropy source for probabilistic inference. Coupled with a PCM-based crossbar for in-memory multiplication enables a form of photonic probabilistic computing that directly supports BNNs.

The specific challenges we faced were twofold. First, the noise needed to be statistically controllable; otherwise, the stochasticity of chaotic light would remain an unusable disturbance. We addressed this by exploiting the intrinsic mean–variance relationship of Bose–Einstein statistics and developing a controllable-noise encoding scheme, which allowed the variance of activations to be tuned within defined bounds. Second, the BNN had to be adapted to the restrictions of the hardware, including limited precision, a constant base noise floor, strictly positive signals, and the small number of available weight cells. Prior to this work, it was unclear whether such a constrained BNN would be feasible at all.

We addressed these challenges through a series of design choices, which were

consolidated in a probabilistic average-pooling operator implemented in Pyro. This operator realizes randomness at the level of activations rather than weights, incorporates QAT to align the network representation with the effective bit-width of the hardware, and uses a Gaussian approximation to the Bose–Einstein PDFs for efficient digital training. As a result, the developed operator captures the essential characteristics of the photonic prototype in a compact and trainable form, making it the central enabler to train BNNs for uncertainty-aware inference on photonic hardware.

From this prototype, several key findings emerged. Chaotic light can be made into a tunable and practically useful entropy source. Photonic BNNs achieved both high in-domain accuracy and clear separation of in-domain and out-of-domain data on a 9-class MNIST task, demonstrating reliable uncertainty estimation. Most importantly, the study shows that careful algorithm–hardware co-design enables probabilistic inference even under the unconventional constraints of photonic devices.

Looking ahead, this proof-of-concept suggests several promising directions for scaling photonic probabilistic computing. Larger crossbars and more complex operators, such as convolutions or attention mechanisms, could extend the scope beyond simple pooling layers. Closer integration of photonic and electronic domains may help to reduce interface bottlenecks and enable more comprehensive accelerator designs. More broadly, the results indicate that noise—long regarded as a limitation of analog computing—can be reframed as a computational resource, opening the possibility of future accelerators that combine speed, energy efficiency, and uncertainty awareness.

11

Conclusion and Outlook

This work addresses two central challenges in modern machine learning: (i) how to enable *resource-efficient inference* of neural networks on resource-constrained or emerging hardware platforms, and (ii) how to extend classical models towards *trustworthy predictions* by incorporating uncertainty estimation.

To this end, we study both *deterministic and probabilistic neural networks*: classical deep neural networks, where efficiency is the primary goal, and their probabilistic counterpart, Bayesian neural networks, which explicitly incorporate uncertainty quantification. For both paradigms, we pursue two strategies to improve efficiency. On the one hand, we reduce computational costs through algorithmic optimizations and code generation targeting digital processors. On the other hand, we explore analog accelerator prototypes, where operations can be realized more cheaply at the physical level, but robustness against nonidealities and tight hardware–algorithm co-design are essential.

On *digital processors for deterministic DNNs*, we advanced automatic model compression by combining latency measurements with sensitivity analysis to guide per-layer quantization and pruning. This work culminated in the *Galen* framework, which establishes how guided policies can unify algorithmic compres-

sion and hardware-aware deployment, thereby making DNN inference practical even on resource-constrained embedded devices.

For *analog accelerators executing deterministic neural networks*, we investigated how nonidealities such as device-level noise affect inference. We developed simplified hardware models that were validated on BSS-2 prototypes and revealed partially surprising imperfections, such as non-associativity in dot products. These models enabled targeted training adaptations and also supported the development of diagnostic tools: in particular, Walking Noise proved highly effective at exposing layer-specific sensitivity. Building on these insights, we proposed hardening techniques to sustain accuracy under perturbations. In particular, Variance-Aware Noisy Training emerged as a simple yet impactful method, showing that neural networks can tolerate a wide range of hardware imperfections if these are explicitly incorporated during training.

Turning to probabilistic models aimed at capturing uncertainty, namely *BNNs on digital processors*, we addressed the high computational cost of Bayesian inference. This work developed scalable approximation and implementation techniques that enable efficient BNN inference on embedded hardware. We began by analyzing BNNs and their core inference methods, MCMC and SVI, evaluating their scalability and uncertainty estimation quality, and revealing how activation functions critically affect convergence and predictive behavior. Building on these insights, we introduced the Probabilistic Forward Pass, an extreme approximation of stochastic variational inference that propagates distributions in closed form, thereby avoiding sampling and repeated forward passes. Within this work, we extended PFP with compiler-based operator support and demonstrated its practical deployment on embedded processors. In addition, we investigated ensemble-based approaches such as Monte Carlo Dropout, Deep Ensembles, and Repulsive Last-Layer Ensembles, which integrate naturally with existing machine learning frameworks. Together, these methods illustrate how algorithmic approximations and compiler-based deployment can make uncertainty-aware inference feasible on embedded systems.

Finally, for *BNNs on analog accelerators*, we combined probabilistic modeling with emerging photonic hardware. Here, we demonstrated how chaotic light can act as a physical entropy source, enabling ultrafast probabilistic computation and uncertainty estimation. This contribution establishes photonic processors as promising candidates for native BNN inference, where hardware noise is

harnessed as a computational resource rather than treated as an obstacle.

Taken together, these contributions establish a unified perspective: resource-efficiency and trustworthiness can be advanced hand-in-hand across deterministic and probabilistic paradigms, by aligning algorithmic methods, compiler technology, and hardware-aware training for both digital and analog accelerators.

11.1 Discussion of Key Insights

The following discussion distills the key insights that emerged across the presented studies. Rather than reiterating individual contributions, it focuses on the overarching principles and trade-offs that shape the design space of resource-efficient inference.

A first overarching insight is that efficiency cannot be achieved by algorithms in isolation. While compression techniques such as pruning and quantization provide the levers, their impact depends critically on the way networks are mapped to hardware. Latency measurements and sensitivity analysis proved far more reliable than simple cost metrics, underscoring that the operating point of compressed models must be determined by actual hardware behavior. The broader lesson is that efficiency emerges from aligning algorithmic strategies with hardware mapping, while balancing the inevitable trade-off between accuracy and efficiency that bounds how far compression can be pushed.

On analog accelerators, the central challenge is not computational cost but robustness against nonidealities. The key insight from modeling analog hardware is that neural networks can tolerate substantial imperfections—noise, drift or nonlinearities—if these effects are systematically embedded into the training process. Diagnostic tools such as Walking Noise proved effective in revealing sensitivity patterns, guiding adaptations, and, most importantly, in clarifying how DNNs learn to tolerate noise. A recurring theme was that robustness is best achieved through *training with increasing complexity*, gradually exposing models to more realistic noise levels or hardware representations—an idea closely related to *curriculum learning* [222], where tasks are structured from simple to complex to facilitate stable and effective learning. VANT exemplifies this principle: although simple in form, it provided a general and effective strategy for improving robustness in noisy settings. Taken together, these findings suggest

that robustness on analog hardware depends less on eliminating imperfections and more on embracing them as part of the training process.

For probabilistic models, the central challenge lies in reconciling uncertainty estimation with computational feasibility on digital processors. Sampling-based inference remains the quality benchmark, but its cost renders it impractical at scale, and even scalable variational methods, while viable with KL annealing, continue to exhibit strong sensitivity to hyperparameters. The broader lesson is that Bayesian neural networks, unlike their deterministic counterparts, remain bounded not only by efficiency constraints but also by the stability of their training dynamics.

Beyond cost, however, two further insights proved critical for uncertainty quality itself. First, the choice of activation function has a surprisingly strong impact: our experiments showed that different nonlinearities can dramatically alter both the calibration and the out-of-distribution detection quality of BNNs, and that the optimal choice is often task dependent.

Second, while single-mode variational inference remains among the most scalable approaches to Bayesian inference, its mean-field formulation imposes inherent limitations when confronted with the complex structure of real posterior distributions. Empirical and theoretical studies have shown that the true posteriors of BNNs are typically multi-modal, heavy-tailed, and strongly correlated [42], [77], causing mean-field approximations to underestimate uncertainty or to miss multiple plausible explanations of the data—an effect that may also contribute to their pronounced sensitivity to hyperparameters and architectural design choices. Nevertheless, as demonstrated in Chapter 7, scalability can also provide a distinct advantage: by enabling the training of more expressive and deeper architectures, SVI allows representational power to partly offset the simplifying assumptions inherent to the inference scheme. Hence, although SVI may not fully capture the multi-modality of complex posteriors, suitably designed architectures can alleviate these shortcomings and still yield uncertainty estimates of high practical relevance at comparatively low computational cost.

Taken together, these observations illustrate that the practical value of SVI extends beyond its theoretical limitations. While mean-field inference restricts posterior expressiveness, its efficiency renders it particularly suitable for deployment on energy- and memory-constrained platforms. However, when moving from general-purpose accelerators to embedded systems, even relatively

lightweight SVI variants become computationally demanding. To address this, two complementary yet fundamentally distinct strategies are examined: the Probabilistic Forward Pass as a minimalist variational approximation, and RLLEs as ensemble-based alternatives offering comparable uncertainty quality at minimal computational cost.

The Probabilistic Forward Pass remains within the variational framework and, by design, is single-modal. It provides a principled, calibration-efficient approximation but cannot capture the full expressiveness of multi-modal posteriors, and its specialized operations are not natively supported by current compiler stacks or hardware backends, which necessitated the implementation of a dedicated operator library within TVM. By contrast, RLLEs explicitly aim to preserve the multi-modality characteristic of Deep Ensembles through their repulsive multi-head formulation. They leverage dense operators that are already highly optimized in modern toolchains, enabling excellent speed and scalability, albeit at the expense of increased calibration sensitivity. Empirical comparisons reinforced this contrast: while RLLEs consistently achieved lower latency and stronger out-of-distribution detection performance, they required careful tuning of repulsive samples, whereas PFP remained more stable but fundamentally limited in posterior expressiveness and more challenging to deploy efficiently.

The broader principle is that uncertainty-aware inference on digital platforms is governed by a multi-dimensional trade-off: efficiency, calibration effort, posterior expressiveness, and theoretical grounding rarely align within a single method. PFP and RLLEs thus occupy complementary positions in this design space, exemplifying the spectrum of viable pathways for embedding uncertainty estimation into resource-constrained systems.

A further insight concerns the asymmetry between training and inference costs across Bayesian methods. At one extreme, Monte Carlo Dropout requires almost no additional probabilistic overhead during training, but typically demands many forward passes at inference, in practice exceeding the cost of MCMC or SVI-based BNNs. At the other extreme, HMC entails substantial training cost, yet inference can be reduced to evaluating a small, carefully chosen subsample of the chain. This can be interpreted as a theoretically grounded, well-selected ensemble. The comparison highlights that training and inference costs are not necessarily aligned. A method considered simple at training time, such as MCDO, can become unexpectedly costly during inference, while a computationally expensive

training method, such as HMC, can yield relatively cheap predictions once samples are available. Which approach is preferable therefore depends on the specific constraints—whether efficiency during training or efficiency during deployment is the dominant bottleneck—underscoring that Bayesian methods must be evaluated across the entire pipeline rather than in a single phase.

When combining probabilistic models with analog accelerators, a striking insight emerges: hardware noise, traditionally seen as a limitation, can be harnessed as a useful source of stochasticity. In particular, photonic accelerators based on chaotic light demonstrate that entropy can be generated at extremely high speed directly in hardware, enabling native probabilistic computation driven by intrinsically rapid and low-cost photonic sampling rather than pseudo-random number generation. This shifts the narrative from noise mitigation to noise exploitation, and highlights that in the analog domain, imperfections are not only unavoidable but can be productively integrated into the computational model.

The broader lesson here is the necessity of hardware–algorithm co-design. The ability to use chaotic light as a physical entropy source did not arise from hardware in isolation, but from jointly adapting probabilistic models, training procedures, and experimental protocols to the specific statistical properties of the photonic device. Without such integration, leveraging noise for uncertainty estimation would not have been feasible.

Taken together, these findings point to a clear principle. Analog photonic devices illustrate that probabilistic inference need not be an add-on implemented in software but can emerge directly from the physics of computation. This suggests that future hardware platforms may move beyond deterministic acceleration of neural networks and instead provide native support for uncertainty estimation, blurring the boundary between algorithm and hardware.

Taken together, these findings reveal two unifying themes that extend across deterministic and probabilistic models as well as digital and analog hardware. First, efficiency and robustness are consistently enabled by *training with increasing complexity*: gradually introducing noise, hardware realism, or regularization such as KL annealing proved essential for stability and generalization. Second, genuine progress requires *algorithm–hardware co-design*, where compression strategies, probabilistic approximations, and robustness techniques are explicitly matched to the constraints and opportunities of the target platform. These principles illustrate that resource-efficiency and uncertainty-awareness cannot

be addressed by isolated algorithmic innovation alone, but must emerge from a joint optimization of models, training schemes, and hardware execution.

11.2 Limitations

While the results of this work demonstrate the feasibility and potential of resource-efficient and uncertainty-aware inference across digital and analog platforms, several limitations must be acknowledged.

First, the scope of the models and datasets is deliberately restricted. Most experiments rely on relatively small networks and controlled benchmarks. The use of small models is motivated by both methodological and hardware constraints: for Bayesian studies, compact architectures are required to make baselines such as HMC with NUTS and SVI computationally feasible, while for analog hardware, small networks are necessary to make underlying effects observable and to reflect the limitations of current prototypes. Although this enables careful experimentation and clear attribution of effects, it also limits the direct transfer of results to modern large-scale architectures, such as deep convolutional neural networks and transformers.

Second, the conclusions regarding automatic model compression remain tied to specific hardware architectures. The experimental results presented here focus on embedded ARM CPUs, and the measured performance gains therefore reflect this class of processors in particular. While the absolute trade-offs observed are hardware-specific, the underlying approach is generic: built on the TVM compiler stack, *Galen* generalizes across the many architectures supported by TVM and thus applies to a wide variety of accelerators beyond those explicitly studied in this work.

Third, the robustness studies on analog accelerators relied on simplified hardware models to keep training efficient. This represents a trade-off: highly accurate models can capture device behavior more closely, but their complexity makes them too slow to be included in training loops, whereas simplified models strike a balance between realism and efficiency. These models, validated against BSS-2 prototypes, proved sufficient to reveal important nonidealities such as non-associativity in dot products and enabled targeted training adaptations.

Complementing this, Walking Noise provided a powerful diagnostic that deepened our understanding of how noisy training affects learning and robustness. Its

computational expense, however, prevents scaling to modern large architectures, highlighting the need for cheaper yet informative sensitivity metrics in future work.

Fourth, while Bayesian inference methods were advanced in this work, their limitations must be carefully delineated. The Probabilistic Forward Pass is deployable in practice, as we provided a compiler-integrated library for embedded platforms, but its expressiveness remains bounded by the assumptions of SVI, in particular its single-mode nature. RLLEs provide a pragmatic and highly efficient approximation, yet their reliance on calibration with repulsive samples—which are essentially artificial out-of-distribution data points—introduces certain limitations. The quality of the resulting uncertainty estimates depends critically on the expressiveness of these repulsive samples and on how well they resemble true OOD settings. This sensitivity highlights both the promise and the fragility of repulsion-based approximations in practical scenarios.

More broadly, two general limitations of current Bayesian neural networks became apparent. First, their uncertainty quality is highly sensitive to the choice of activation function, with different nonlinearities yielding substantially different calibration and OOD detection behavior. Second, approximating inherently multi-modal posteriors with single-mode methods, as in most variational inference approaches, remains fundamentally limited and highlights the need for richer yet efficient inference techniques.

Finally, the photonic experiments represent proof-of-concept demonstrations. While chaotic light was successfully harnessed as a physical entropy source, scaling these approaches to larger networks and establishing true probabilistic weight representations remain open challenges——but also areas of rapid progress as photonic devices continue to advance in scale and integration.

Recognizing these limitations is essential for contextualizing the contributions of this work. They also provide a natural entry point into the outlook for future research, where the lessons learned here may be extended to larger models, more diverse hardware platforms, and more mature uncertainty estimation techniques. Especially transformer architectures and large language models would be highly interesting to investigate from a principled point of view, yet they also pose a substantial challenge for the probabilistic machine learning community.

11.3 Outlook

Acknowledging these challenges, a key direction for future work is to extend the evaluated methods—and Bayesian neural networks more broadly—to large-scale architectures such as transformers, large language models, and modern diffusion-based generative models. The experiments in this work were deliberately restricted to smaller networks to allow controlled evaluation and comparison with high-quality baselines, but scalability remains a central open question in the Bayesian research community. Partial Bayesianization [24], [65] offers one promising path: rather than modeling all parameters probabilistically, only selected layers or subsets of weights carry uncertainty, thereby balancing tractability with uncertainty quality. This idea was explored to through RLLEs, which retain the efficiency of deterministic backbones while enriching the output distribution with repulsive multi-head structures. Still, fundamental questions remain: which components should be Bayesian, what fraction of parameters is sufficient, and how can such structures be identified automatically? For large language models, these challenges are compounded by their sequential autoregressive inference, where uncertainty must be propagated token by token and efficiency is dominated by long-context predictions. Tackling these questions would make probabilistic methods more tractable at scale and move them closer to practical deployment in the architectures that define modern deep learning.

The role of activation functions provides another promising avenue. Our results showed that uncertainty quality in Bayesian neural networks depends strongly on the chosen nonlinearity, with performance varying substantially across tasks. Rather than selecting from a fixed set of functions, future work could focus on learning activation functions directly, for instance with approaches inspired by Kolmogorov–Arnold Networks [5]. Such methods could produce activation functions adapted to both model and task, and in the longer term may open the door to Bayesian architecture search, offering new insight into what architectural properties make networks naturally suited for probabilistic inference.

At the algorithmic level, opportunities arise to overcome the single-mode limitations of variational inference. The Probabilistic Forward Pass provides an efficient route to closed-form propagation, but its expressiveness remains constrained. A promising extension is to combine it with ensemble methods, enriching their multi-modal character with local variational information. Similarly,

Repulsive Last-Layer Ensembles demonstrated high efficiency, but their reliance on repulsive sample calibration remains a central limitation. A way to train these ensembles without artificial out-of-distribution data would be highly desirable, yet the appropriate approach remains an open question. Maybe alternative strategies, such as independently training ensemble heads or designing more natural forms of repulsion, could help to reduce calibration sensitivity and improve robustness.

On the model compression side, the *Galen* framework illustrates how automatic compression policies can be aligned with measured hardware performance, but also suggests natural extensions. Incorporating per-layer latency measurements and low-level hardware information would provide even more detailed feedback to guide compression decisions. Beyond embedded CPUs and GPUs, extending the framework to platforms such as FPGAs or ASICs will require improved cost models, since direct hardware-in-the-loop profiling becomes infeasible at scale. Such developments would broaden the reach of automatic compression and further integrate algorithmic techniques with architectural considerations.

In the area of robustness, diagnostics such as Walking Noise provided unique insight into layer-wise sensitivity and helped to understand how noisy training improves robustness. Its computational cost, however, makes it unsuitable for large architectures. What is needed in general is a cheap but informative sensitivity metric. Two promising directions emerge: the first seeks to conserve the overall noise budget and learn its optimal distribution across layers, thereby capturing relative sensitivity within a single training run rather than through repeated evaluations. The second applies KL-divergence-based perturbations, analogous to the sensitivity analysis employed in *Galen*, where the divergence between perturbed and baseline predictions serves as an efficient proxy for layer-wise robustness. Developing such metrics would combine interpretability with scalability, making robustness analysis more practical for modern neural networks.

For analog accelerators, the statistical properties of device noise align naturally with stochastic variational inference, since many noise processes follow Gaussian or other well-characterized parametric distributions. This makes variational methods a well-matched candidate for probabilistic inference on analog hardware. At the same time, enriching SVI with ensemble ideas could help to overcome its single-mode limitations, enabling richer multi-modal posteriors while retaining the efficiency advantages that make SVI attractive in this setting.

More broadly, progress in Bayesian inference may depend on building bridges between methods. High-quality MCMC posteriors can serve as a reference or teacher model to regularize or initialize SVI, while priors learned by SVI can accelerate MCMC by providing informed starting points. First results suggest that knowledge distillation can be effective in transferring information from MCMC to SVI BNNs. Exploring such transfers opens a wider design space in which different inference methods complement each other, combining the quality of MCMC with the scalability of SVI.

Finally, photonic accelerators represent one of the most promising frontiers for probabilistic machine learning. The demonstrations in this work established that chaotic light can serve as a physical entropy source, providing a fast and high-quality basis for native probabilistic computation. Future generations of photonic hardware are expected to offer greater scale, finer controllability, and tighter integration with mainstream machine learning frameworks. At the same time, they will raise new challenges, including the tolerance of accumulating base noise in deeper architectures and the development of training methods that can adapt to device-specific noise processes. Sustained progress in this area will depend on close hardware–algorithm co-design, ensuring that advances in photonic devices and probabilistic modeling reinforce each other as the technology matures.

Concluding Remarks

The questions that motivated this work lie at the core of modern machine learning: how to make neural networks more resource-efficient, and how to make their predictions more trustworthy. These challenges are far from solved and will continue to occupy the field for years to come. Yet, the studies presented here provide concrete steps toward these goals, demonstrating how efficiency and robustness can be advanced together when hardware, compilers, and algorithms are considered as parts of a unified system.

A recurring theme throughout this work is that the most effective strategies tend to embrace, rather than avoid, complexity and imperfection. Training with increasing realism—through progressive noise schedules, hardware-model refinement, or KL annealing—consistently enabled models to adapt and generalize under challenging conditions. Similarly, hardware–algorithm co-design proved

essential: performance improvements arose not merely from better models, but from aligning those models with the computational structure and operational constraints of the hardware that realizes them. These insights suggest a broader view of neural network design in which computation, approximation, and physical realization are treated as a single, integrated system.

Looking forward, the principles established here provide a foundation for the next generation of machine learning systems—systems that are both efficient and trustworthy, and that blur the boundary between software and hardware. Whether applied to large-scale models or to emerging analog accelerators, the same guiding ideas hold: leverage the structure of the hardware, expose imperfections during training, and design algorithms that learn to thrive within their physical constraints. By following these principles, the field moves closer to neural networks that not only compute efficiently, but also reason with confidence about what they do not know.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Professor Dr. Holger Fröning. I have been very fortunate to benefit from his guidance over the past years and from the opportunity to work on several novel and exciting research topics. At every stage, Holger provided excellent advice, generous support, and sincere as well as constructive feedback. I am deeply indebted to him for his continuous encouragement and for the many opportunities he offered throughout my doctoral studies.

I would also like to thank my co-advisor, Professor Dr. Franz Pernkopf, for his professional expertise and invaluable advice. Franz enabled me to collaborate with his group members Dr. Wolfgang Roth and Sophie Steger, whose outstanding research and insightful discussions greatly enriched this work and deepened my understanding of probabilistic machine learning.

It has been a true pleasure to work within the *HAWAII* research group. I am sincerely grateful to Dr. Günther Schindler and Dr. Lorenz Braun for their mentorship, their openness to share expertise, and for the many insightful discussions that shaped the early stages of my research. I would also like to thank Andrea Seeger, Dr. Felix Zahn, Dr. Vahdaneh Kiani, and Dr. Jonas Dann for their valuable feedback, friendly collaboration, and for creating an inspiring and supportive research atmosphere during this time. Over the years, our group evolved, and I feel genuinely fortunate to now work side by side with my current colleagues—Hendrik Borras, Xiao Wang, Daniel Barley, Dr. Kazem Shekofteh, Aleksandra Poreba, Alexandra Stehle, and Robin Janssen. The close collaboration, open exchange of ideas, and the joyful daily interactions within this team have made my recent years both deeply rewarding and truly enjoyable. I am especially thankful to Hendrik for his reliability and inspiring spirit in our joint projects.

Throughout my time at the institute, I had the pleasure of collaborating with several excellent students—Lisa Kuhn, Torben Krieger, Falk Selker, Christian Simonides, and Jonathan Bernhard—whose master's theses contributed significantly to this dissertation. Their enthusiasm and commitment made these collaborations both inspiring and enjoyable.

I gratefully acknowledge the financial support provided under the scope of the COMET program within the K2 Center "Integrated Computational Material, Process and Product Engineering (IC-MPPE)" (Project No. 886385). This program is funded by the Austrian Federal Ministries for Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK) and for Labour and Economy (BMAW), represented by the Austrian Research Promotion Agency (FFG), as well as by the federal states of Styria, Upper Austria, and Tyrol.

Finally, I would like to express my heartfelt gratitude to my family and friends for their unwavering support, patience, and belief in me throughout this demanding and exciting journey. Your encouragement has been a constant source of motivation.

Declaration of AI Usage

This declaration is made in accordance with Heidelberg University's guidelines on the responsible and transparent use of generative artificial intelligence in academic writing.

In the preparation of this dissertation, large language model (LLM)–based generative AI tools—specifically *ChatGPT-5* (OpenAI, San Francisco, CA, USA)—were employed solely to support linguistic refinement and the enhancement of readability, as well as to assist in the formulation and structuring of the text. All AI-assisted passages have been critically reviewed, revised, and approved by the author.

All scholarly and scientific contributions—including the conception of ideas, development of methodology, design and execution of experiments, data analyses, and formulation of conclusions—are entirely the author's own work, unless explicitly stated otherwise.

Bernhard Klein
October 2025

Acronyms

ADC	analog-to-digital converter.
AMC	AutoML for Model Compression.
ARM	Advanced RISC Machines.
ASE	amplified spontaneous emission.
ASIC	Application-Specific Integrated Circuit.
AU	aleatoric uncertainty.
AUROC	area under the receiver operating characteristic curve.
BN	batch normalization.
BNN	Bayesian neural network.
BOPs	bit operations.
BS	batch size.
BSS-2	BrainScaleS-2.
CMOS	complementary metal–oxide–semiconductor.
CNN	convolutional neural network.
CPU	Central Processing Unit.
DDPG	Deep Deterministic Policy Gradient.
DE	Deep Ensemble.
DNN	deep neural network.
ECE	expected calibration error.
ELBO	evidence lower bound.
ESS	effective sample size.
EU	epistemic uncertainty.
EUC	Epistemic Uncertainty Coefficient.

fLL-POVI	Function-Space Last-Layer Particle-Optimization Variational Inference.
FLOPs	floating-point operations.
FPGA	Field-Programmable Gate Array.
FPR	false positive rate.
GPU	Graphics Processing Unit.
GSC	Google Speech Commands.
GST	germanium–antimony–telluride.
HAQ	Hardware-Aware Quantization.
HIL	hardware-in-the-loop.
HMC	Hamiltonian Monte Carlo.
ID	in-distribution.
IoT	Internet of Things.
IR	intermediate representation.
ISA	Instruction Set Architecture.
JIT	just-in-time.
KAN	Kolmogorov–Arnold Network.
KDE	Kernel Density Estimation.
KL	Kullback–Leibler divergence.
LL-POVI	Last-Layer Particle-Optimization Variational Inference.
LLM	large language model.
LUT	lookup table.
MAC	multiply–accumulate.
MCDO	Monte Carlo Dropout.
MCMC	Markov chain Monte Carlo.
MI	mutual information.
ML	machine learning.
MLIR	Multi-Level Intermediate Representation.
MLP	multi-layer perceptron.
MSE	mean squared error.
MZI	Mach–Zehnder interferometer.
NAS	neural architecture search.
NLL	negative log-likelihood.
NLP	natural language processing.

NN	neural network.
NPU	Neural Processing Unit.
NUTS	No-U-Turn Sampler.
OOD	out-of-distribution.
OTA	operational transconductance amplifier.
PCM	phase-change material.
PDF	probability density function.
PFP	Probabilistic Forward Pass.
POVI	Particle-Optimization Variational Inference.
PPL	Probabilistic Programming Language.
QAT	quantization-aware training.
rAUC	relative area under the accuracy–noise curve.
RDE	Repulsive Deep Ensemble.
ReLU	Rectified Linear Unit.
RLL-POVI	Repulsive Last-Layer Particle-Optimization Variational Inference.
RLLE	Repulsive Last-Layer Ensemble.
RNN	recurrent neural network.
RRAM	resistive random-access memory.
SAM	Sharpness-Aware Minimization.
SDE	stochastic differential equation.
SGD	stochastic gradient descent.
SIMD	Single Instruction Multiple Data.
SME	softmax entropy.
SoC	System-on-Chip.
STD	standard deviation.
STE	straight-through estimator.
SVI	stochastic variational inference.
TE	tensor expression.
THz	terahertz.
TPR	true positive rate.
TPU	Tensor Processing Unit.
TVM	Tensor Virtual Machine.
VANT	Variance-Aware Noisy Training.
VGG	Visual Geometry Group.

VI	variational inference.
WDM	wavelength-division multiplexing.
WRN	Wide Residual Network.
XLA	Accelerated Linear Algebra.

References

[1] F. Brückerhoff-Plückelmann, A. P. Ovvyan, A. Varri, H. Borras, B. Klein, L. Meyer, C. D. Wright, H. Bhaskaran, G. S. Syed, A. Sebastian, H. Fröning, and W. Pernice, "Probabilistic photonic computing for ai", *Nature Computational Science*, May 2025, DOI: 10.1038/s43588-025-00800-1.

[2] S. Jiménez, M. Jürgens, and W. Waegeman, *Why machine learning models fail to fully capture epistemic uncertainty*, 2025, arXiv: 2505.23506.

[3] B. Klein, F. Selker, H. Borras, S. Steger, F. Pernkopf, and H. Fröning, "Accelerated execution of bayesian neural networks using a single probabilistic forward pass and code generation", *under review at ACM Transactions on Architecture and Code Optimization*, 2025.

[4] L. Kuhn, B. Klein, and H. Fröning, "On the Non-Associativity of Analog Computations", in *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, R. Meo and F. Silvestri, Eds., Springer Nature Switzerland, Sep. 2025, pp. 183–195, DOI: https://doi.org/10.1007/978-3-031-74643-7_15, arXiv: 2309.14292.

[5] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljacic, T. Y. Hou, and M. Tegmark, "KAN: Kolmogorov-arnold networks", in *International Conference on Learning Representations (ICLR)*, 2025, https://openreview.net/forum?id=Ozo7qJ5vZi.

[6] C. Simonides, "Efficient ensemble-based bayesian neural networks for depth regression", M.S. thesis, Heidelberg University, 2025, DOI: 10.11588/heidok.00037231.

[7] X. Wang, H. Borras, B. Klein, and H. Fröning, "On hardening dnns against noisy computations", in *HiPEAC, Workshop on Accelerated Machine Learning (AccML)*, 2025, arXiv: 2501.14531, https://accml.dcs.gla.ac.uk/papers/2025/7th_AccML_paper_1.pdf.

[8] X. Wang, H. Borras, B. Klein, and H. Fröning, "Variance-aware noisy training: Hardening dnns against unstable analog computations", in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*, 2025, arXiv: 2503.16183.

[9] H. Borras, B. Klein, and H. Fröning, "Walking noise: On layer-specific robustness of neural architectures against noisy computations and associated characteristic learning dynamics", in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKKD)*, 2024, arXiv: 2212.10430, https://doi.org/10.1007/978-3-031-70359-1_3.

[10] F. Brückerhoff-Plückelmann, H. Borras, B. Klein, A. Varri, M. Becker, J. Dijkstra, M. Brückerhoff, C. D. Wright, M. Salinga, H. Bhaskaran, B. Risse, H. Fröning, and W. Pernice, "Probabilistic photonic computing with chaotic light", *Nature Communications*, vol. 15, no. 1, p. 10 445, Dec. 2024, https://doi.org/10.1038/s41467-024-54931-6.

[11] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, *Awq: Activation-aware weight quantization for llm compression and acceleration*, 2024, arXiv: 2306.00978, https://arxiv.org/abs/2306.00978.

[12] W. Roth, G. Schindler, B. Klein, R. Peharz, S. Tschiatschek, H. Fröning, F. Pernkopf, and Z. Ghahramani, "Resource-Efficient Neural Networks for Embedded Systems", *Journal of Machine Learning Research (JMLR)*, vol. 25, no. 50, pp. 1–51, 2024, arXiv: 2001.03048, http://jmlr.org/papers/v25/18-566.html.

[13] S. Steger, C. Knoll, B. Klein, H. Fröning, and F. Pernkopf, "Function space diversity for uncertainty prediction via repulsive last-layer ensembles", in *ICML 2024 Workshop on Structured Probabilistic Inference & Generative Modeling*, 2024, https://openreview.net/forum?id=FbMN9HjgHI.

[14] H. Awano and M. Hashimoto, "B2n2: Resource efficient bayesian neural network accelerator using bernoulli sampler on fpga", *Integration*, vol. 89, pp. 1–8, 2023, DOI: https://doi.org/10.1016/j.vlsi.2022.11.005.

[15] D. Bonnet, T. Hirtzlin, A. Majumdar, T. Dalgaty, E. Esmanhotto, V. Meli, N. Castellani, S. Martin, J.-F. Nodin, G. Bourgeois, J.-M. Portal, D. Querlioz, and E. Vianello, "Bringing uncertainty quantification to the extreme-edge with memristor-based bayesian neural networks", *Nature Communications*, vol. 14, no. 1, p. 7530, Nov. 2023, DOI: 10.1038/s41467-023-43317-9.

[16] H. Borras, B. Klein, and H. Fröning, "Walking Noise: Understanding Implications of Noisy Computations on Classification Tasks", in *HiPEAC Conference, Workshop on Accelerated Machine Learning (AccML)*, Jan. 2023, arXiv: 2212.10430, https://accml.dcs.gla.ac.uk/papers/2023/5th_AccML_paper_2.pdf.

[17] S. Feng, B. Hou, H. Jin, W. Lin, J. Shao, R. Lai, Z. Ye, L. Zheng, C. H. Yu, Y. Yu, and T. Chen, "Tensorir: An abstraction for automatic tensorized program optimization", in *International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS,

Vancouver, BC, Canada: Association for Computing Machinery, 2023, pp. 804–817, DOI: 10.1145/3575693.3576933.

[18] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "OPTQ: Accurate quantization for generative pre-trained transformers", in *International Conference on Learning Representations (ICLR)*, 2023, arXiv: 2210.17323.

[19] A. H. Gadhikar, S. Mukherjee, and R. Burkholz, "Why random pruning is all we need to start sparse", in *Proceedings of Machine Learning Research (PMLR)*, vol. 202, 2023, https://proceedings.mlr.press/v202/gadhikar23a.html.

[20] N. P. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles, C. Young, X. Zhou, Z. Zhou, and D. Patterson, "Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings", *Proceedings of international symposium on computer architecture*, 2023, DOI: https://doi.org/10.1145/3579371.3589350.

[21] T. Krieger, B. Klein, and H. Fröning, "Towards Hardware-Specific Automatic Compression of Neural Networks", *AAAI Conference on Artificial Intelligence, International Workshop on Practical Deep Learning in the Wild*, Feb. 2023, Best Paper Award, DOI: 10.48550/arXiv.2212.07818, arXiv: 2212.07818.

[22] R. Lai *et al.*, "Relax: Composable abstractions for end-to-end dynamic machine learning", 2023, arXiv: 2311.02103.

[23] K. P. Murphy, *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023, http://probml.github.io/book2.

[24] M. Sharma, S. Farquhar, E. Nalisnick, and T. Rainforth, "Do bayesian neural networks need to be fully stochastic?", in *International Conference on Artificial Intelligence and Statistics*, F. Ruiz, J. Dy, and J.-W. van de Meent, Eds., vol. 206, PMLR, 25–27 Apr 2023, pp. 7694–7722, https://proceedings.mlr.press/v206/sharma23a.html.

[25] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, *Llama: Open and efficient foundation language models*, 2023, arXiv: 2302.13971, https://arxiv.org/abs/2302.13971.

[26] X. Wu, P. Paramasivam, and V. Taylor, "Autotuning apache tvm-based scientific applications using bayesian optimization", in *SC Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, Association for Computing Machinery, 2023, pp. 29–35, DOI: 10.1145/3624062.3626079.

[27] J. van Amersfoort, L. Smith, A. Jesson, O. Key, and Y. Gal, "On feature collapse and deep kernel learning for single forward pass uncertainty", 2022, arXiv: 2102.11409.

[28] L. V. Jospin, H. Laga, F. Boussaid, W. Buntine, and M. Bennamoun, "Hands-on bayesian neural networks—a tutorial for deep learning users", *IEEE Computational Intelligence Magazine*, vol. 17, no. 2, pp. 29–48, 2022, DOI: 10.1109/MCI.2022.3155327.

[29] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, 2022, arXiv: 1312.6114, https://arxiv.org/abs/1312.6114.

[30] F. Li, B. Liu, X. Wang, B. Zhang, and J. Yan, *Ternary weight networks*, 2022, arXiv: 1605.04711, https://arxiv.org/abs/1605.04711.

[31] J. Mukhoti, A. Kirsch, J. van Amersfoort, P. H. S. Torr, and Y. Gal, "Deep Deterministic Uncertainty: A Simple Baseline", Tech. Rep., Jan. 2022, arXiv: 2102.11582.

[32] J. Shao, X. Zhou, S. Feng, B. Hou, R. Lai, H. Jin, W. Lin, M. Masuda, C. H. Yu, and T. Chen, "Tensor program optimization with probabilistic programs", in *Advances in Neural Information Processing Systems (NeurIPS)*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, Curran Associates, Inc., 2022, pp. 35 783–35 796, arXiv: 2205.13603.

[33] P. Tempczyk, K. Smoczyński, P. Smolenski-Jensen, and M. Cygan, *One simple trick to fix your bayesian neural network*, 2022, arXiv: 2207.13167, https://arxiv.org/abs/2207.13167.

[34] P. Villalobos, J. Sevilla, T. Besiroglu, L. Heim, A. Ho, and M. Hobbhahn, *Machine learning model sizes and the parameter gap*, 2022, arXiv: 2207.02852, https://arxiv.org/abs/2207.02852.

[35] C. Baskin, N. Liss, E. Schwartz, E. Zheltonozhskii, R. Giryes, A. M. Bronstein, and A. Mendelson, "Uniq: Uniform noise injection for non-uniform quantization of neural networks", in *Proceedings of the International Conference on Machine Learning (ICML)*, 2021, pp. 684–693, DOI: 10.1145/3444943.

[36] B. Cramer, S. Billaudelle, S. Kanya, A. Leibfried, A. Grübl, V. Karasenko, C. Pehle, K. Schreiber, Y. Stradmann, J. Weis, J. Schemmel, and F. Zenke, *Surrogate gradients for analog neuromorphic computing*, 2021, arXiv: 2006.07239, https://arxiv.org/abs/2006.07239.

[37] F. D'Angelo and V. Fortuin, "Repulsive deep ensembles are bayesian", in *Advances in Neural Information Processing Systems (NeurIPS)*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34, Curran Associates, Inc., 2021, pp. 3451–3465, https://proceedings.neurips.cc/paper_files/paper/2021/file/1c63926ebcabda26b5cdb31b5cc91efb-Paper.pdf.

[38] E. Daxberger, A. Kristiadi, A. Immer, R. Eschenhagen, M. Bauer, and P. Hennig, "Laplace redux-effortless Bayesian deep learning", *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 20 089–20 103, 2021, https://proceedings.neurips.cc/paper/2021/hash/a7c9585703d275249f30a088cebba0ad-Abstract.html.

[39] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, "Sharpness-aware minimization for efficiently improving generalization", in *International Conference on Learning Representations (ICLR)*, 2021, arXiv: 2010.01412.

[40] J. Gawlikowski, C. R. Tassi, M. Ali, J. Lee, M. Humt, J. Feng, A. Kruspe, R. Triebel, P. Jung, R. Roscher, *et al.*, "A survey of uncertainty in deep neural networks", *Artificial Intelligence Review*, vol. 56, no. 1, pp. 151–243, 2021, DOI: https://doi.org/10.1007/s10462-023-10562-9.

[41] E. Hüllermeier and W. Waegeman, "Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods", *Machine Learning*, vol. 110, no. 3, pp. 457–506, 2021, DOI: 10.1007/s10994-021-05946-3.

[42] P. Izmailov, S. Vikram, M. D. Hoffman, and A. G. Wilson, "What are bayesian neural network posteriors really like?", in *Proceedings of the International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 4629–4640, https://proceedings.mlr.press/v139/izmailov21a.html.

[43] B. Klein, C. Gratl, M. Mücke, and H. Fröning, "Understanding cache boundness of ml operators on arm processors", in *HiPEAC, Workshop on Accelerated Machine Learning (AccML)*, Jan. 2021, arXiv: 2102.00932, https://arxiv.org/abs/2102.00932.

[44] B. Klein, L. Kuhn, J. Weis, A. Emmel, Y. Stradmann, J. Schemmel, and H. Fröning, "Towards addressing noise and static variations of analog computations using efficient retraining", in *Machine Learning and Principles and Practice of Knowledge Discovery in Databases - International Workshops of ECML PKDD 2021, Proceedings Part I*, M. Kamp *et al.*, Eds., ser. Communications in Computer and Information Science, vol. 1524, Springer, 2021, pp. 409–420, DOI: https://doi.org/10.1007/978-3-030-93736-2_32.

[45] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "Mlir: Scaling compiler infrastructure for domain specific computation", in *IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021, pp. 2–14, DOI: 10.1109/CGO51591.2021.9370308.

[46] B. Murmann, "Mixed-signal computing for deep neural network inference", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 1, pp. 3–13, 2021, DOI: https://10.1109/TVLSI.2020.3020286.

[47] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "Efficient large-scale language model training on gpu clusters using megatron-lm", in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, IEEE, 2021, DOI: 10.1145/3458817.3476204.

[48] W. Roth, "Probabilistic methods for resource efficiency in machine learning", Ph.D. dissertation, Graz University of Technology Austria, 2021, https://download.spsc.tugraz.at/thesis/thesis-wroth.pdf.

[49] J. Schemmel, S. Billaudelle, P. Dauer, and J. Weis, "Accelerated analog neuromorphic computing", *Advances in Analog Circuit Design*, pp. 83–102, 2021, DOI: https://doi.org/10.1007/978-3-030-91741-8_6.

[50] A. Vehtari, A. Gelman, D. Simpson, B. Carpenter, and P.-C. Bürkner, "Rank-normalization, folding, and localization: An improved $\hat{R}$ for assessing convergence of MCMC", *Bayesian Analysis*, vol. 16, no. 2, pp. 667–718, 2021, DOI: 10.1214/20-BA1221.

[51] Q. Wan, H. Xia, X. Zhang, L. Wang, S. L. Song, and X. Fu, "Shift-BNN: Highly-efficient probabilistic bayesian neural network training via memory-friendly pattern retrieving", in *IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO, New York, NY, USA: Association for Computing Machinery, 2021, pp. 885–897, DOI: 10.1145/3466752.3480120.

[52] M. van Baalen, C. Louizos, M. Nagel, R. A. Amjad, Y. Wang, T. Blankevoort, and M. Welling, "Bayesian bits: Unifying quantization and pruning", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020, pp. 5741–5752, https://proceedings.neurips.cc/paper_files/paper/2020/hash/3f13cf4ddf6fc50c0d39a1d5aeb57dd8-Abstract.html.

[53] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. V. Le, "Can weight sharing outperform random architecture search? an investigation with tunas", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020, https://openaccess.thecvf.com/content_CVPR_2020/html/Bender_Can_Weight_Sharing_Outperform_Random_Architecture_Search_An_Investigation_With_CVPR_2020_paper.html.

[54] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners", *Advances in neural information processing systems (NeurIPS)*, vol. 33, pp. 1877–1901, 2020, https://proceedings.neurips.cc/paper_files/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[55] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment", in *International Conference on Learning Representations (ICLR)*, 2020, https://openreview.net/forum?id=HylxE1HKwS.

[56] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. Mahoney, and K. Keutzer, "Zeroq: A novel zero shot quantization framework", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 13 166–13 175, DOI: 10.1109/CVPR42600.2020.01318.

[57] Z. Dong, Z. Yao, Y. Cai, D. Arfeen, A. Gholami, M. W. Mahoney, and K. Keutzer, "Hawq-v2: Hessian aware trace-weighted quantization of neural networks", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020, https://proceedings.neurips.cc/paper/2020/hash/d77c703536718b95308130ff2e5cf9ee-Abstract.html.

[58] Y. Du, L. Jing, Y. Li, Z. Ji, H. He, Y. Shen, Q. Tang, S. Wen, and L. Bao, "Exploring the impact of random telegraph noise-induced accuracy loss on resistive ram-based deep neural network", *IEEE Transactions on Electron Devices*, 2020, DOI: 10.1109/TED.2020.3002736.

[59] A. T. Elthakeb, P. Pilligundla, F. S. Mireshghallah, A. Yazdanbakhsh, and H. Esmaeilzadeh, "A reinforcement learning approach for automatic deep quantization of neural networks", *IEEE Micro*, vol. 40, no. 5, pp. 37–45, 2020, DOI: 10.1109/MM.2020.3009475.

[60] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization", in *International Conference on Learning Representations (ICLR)*, 2020, arXiv: 1902.08153.

[61] S. Farquhar, L. Smith, and Y. Gal, "Try depth instead of weight correlations: Mean-field is a less restrictive assumption for variational inference in deep networks", in *Bayesian Deep Learning Workshop At NeurIPS*, 2020, https://bayesiandeeplearning.org/2019/papers/45.pdf.

[62] M. Giacobbe, T. A. Henzinger, and M. Lechner, "How many bits does it take to quantize your neural network?", in *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, Shows non-monotonic robustness vs. bit-width, Springer, 2020, https://link.springer.com/chapter/10.1007/978-3-030-45237-7_5.

[63] S. Gowal, K. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, R. Arandjelović, T. Mann, and P. Kohli, "Uncovering the limits of adversarial training against norm-bounded adversarial examples", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020, arXiv: 2010.03593.

[64] V. Joshi, T. Gokmen, *et al.*, "Accurate deep neural network inference using computational phase-change memory", *Nature Communications*, vol. 11, no. 1, p. 2473, 2020, DOI: 10.1038/s41467-020-16108-9.

[65] A. Kristiadi, M. Hein, and P. Hennig, "Being bayesian, even just a bit, fixes overconfidence in ReLU networks", in *International Conference on Machine Learning (ICML)*, H. Daumé and A. Singh, Eds., ser. Proceedings of Machine Learning Research, vol. 119, PMLR, 13–18 Jul 2020, pp. 5436–5446, https://proceedings.mlr.press/v119/kristiadi20a.html.

[66] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, and Y. Tian, "Channel pruning via automatic structure search", in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2020, pp. 673–679, arXiv: 2001.08565.

[67] N. Liu, X. Ma, Z. Xu, Y. Wang, J. Tang, and J. Ye, "Autocompress: An automatic dnn structured pruning framework for ultra-high compression rates", in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 4876–4883, DOI: https://doi.org/10.1609/aaai.v34i04.5924.

[68] X. Liu, T. Xiao, S. Si, and C.-J. Hsieh, "How does noise help robustness? explanation and exploration under the neural sde framework", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 282–290, https://openaccess.thecvf.com/content_CVPR_2020/html/Liu_How_Does_Noise_Help_Robustness_Explanation_and_Exploration_under_the_CVPR_2020_paper.html.

[69] Q. Lou, F. Guo, L. Liu, M. Kim, and L. Jiang, "Autoq: Automated kernel-wise neural network quantization", in *International Conference on Learning Representations (ICLR)*, 2020, https://openreview.net/forum?id=Hke_0RNKPr.

[70] A. Shekhovtsov, V. Yanush, and B. Flach, "Path sample-analytic gradient estimators for stochastic binary networks", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020, pp. 12 884–12 894, https://proceedings.neurips.cc/paper/2020/hash/96fca94df72984fc97ee5095410d4dec-Abstract.html.

[71] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, *Megatron-lm: Training multi-billion parameter language models using model parallelism*, 2020, arXiv: 1909.08053, https://arxiv.org/abs/1909.08053.

[72] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "How to evaluate deep neural network processors: TOPS/W (alone) considered harmful", *IEEE Solid-State Circuits Magazine*, vol. 12, no. 3, pp. 28–41, 2020, DOI: 10.1109/MSSC.2020.3002142.

[73] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks", 2020, arXiv: 1905.11946, https://arxiv.org/abs/1905.11946.

[74] S. Uhlich, L. Mauch, F. Cardinaux, K. Yoshiyama, J. A. García, S. Tiedemann, T. Kemp, and A. Nakamura, "Mixed precision DNNs: All you need is a good parametrization", in *International Conference on Learning Representations (ICLR)*, 2020, arXiv: 1905.11452.

[75] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han, "Apq: Joint search for network architecture, pruning and quantization policy", in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, DOI: 10.1109/CVPR42600.2020.00215.

[76] Y. Wang, Y. Lu, and T. Blankevoort, "Differentiable joint pruning and quantization for hardware efficiency", in *European Conference on Computer Vision (ECCV)*, 2020, DOI: 10.1007/978-3-030-58526-6_16.

[77] A. G. Wilson and P. Izmailov, "Bayesian deep learning and a probabilistic perspective of generalization", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020, https://proceedings.neurips.cc/paper/2020/hash/f5c1b9e6f3ada9a3b02d00e8f4b3f4ab-Abstract.html.

[78] Y. Yang, E. Frantar, and D. Alistarh, "Bayesft: Bayesian fine-tuning of quantized neural networks", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020, https://arxiv.org/abs/2010.10496.

[79] L. Zheng, C. Jia, M. Sun, Z. Wu, C. H. Yu, A. Haj-Ali, Y. Wang, J. Yang, D. Zhuo, K. Sen, J. E. Gonzalez, and I. Stoica, "Ansor: Generating High-Performance tensor programs for deep learning", in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, USENIX Association, Nov. 2020, pp. 863–879, https://www.usenix.org/conference/osdi20/presentation/%20zheng.

[80] C. Zhou, P. Kadambi, M. Mattina, and P. N. Whatmough, *Noisy machines: Understanding noisy neural networks and enhancing robustness to analog hardware errors using distillation*, 2020, arXiv: 2001.04974, https://arxiv.org/abs/2001.04974.

[81] S. S. Banerjee, Z. T. Kalbarczyk, and R. K. Iyer, "AcMC2: Accelerating markov chain monte carlo algorithms for probabilistic models", in *International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19, New York, NY, USA: Association for Computing Machinery, 2019, pp. 515–528, DOI: 10.1145/3297858.3304019.

[82] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman, "Pyro: Deep universal probabilistic programming", *Journal of Machine Learning Research*, vol. 20, no. 28, pp. 1–6, 2019, http://jmlr.org/papers/v20/18-403.html.

[83] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware", in *International Conference on Learning Representations (ICLR)*, 2019, arXiv: 1812.00332.

[84] V. Camus, A. Valentian, S. Anghel, E. Ismail, and E. Beigne, "A comprehensive crossbar model benchmarking-based evaluation methodology for memristive CNN accelerators", in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2019, pp. 331–336, DOI: 10.1145/3287624.3287705.

[85] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding", in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Association for Computational Linguistics, 2019, pp. 4171–4186, https://aclanthology.org/N19-1423.

[86] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "HAWQ: Hessian aware quantization of neural networks with mixed-precision", in *International Conference on Computer Vision (ICCV)*, 2019, pp. 293–302, https://openaccess.thecvf.com/content_ICCV_2019/html/Dong_HAWQ_Hessian_AWare_Quantization_of_Neural_Networks_With_Mixed-Precision_ICCV_2019_paper.html.

[87] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey", *The Journal of Machine Learning Research (JMLR)*, vol. 20, no. 1, pp. 1997–2017, 2019, https://www.jmlr.org/papers/v20/18-598.html.

[88] S. Fort, H. Hu, and B. Lakshminarayanan, "Deep ensembles: A loss landscape perspective", in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019, https://proceedings.neurips.cc/paper/2019/hash/67e3d64f95417a1e10a2f709b76d61d4-Abstract.html.

[89] M. Havasi, R. Peharz, and J. M. Hernández-Lobato, "Minimal random code learning: Getting bits back from compressed model parameters", in *International Conference on Learning Representations (ICLR)*, 2019, arXiv: 1810.00440.

[90] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for Model Compression and Acceleration on Mobile Devices", Jan. 2019, https://openaccess.thecvf.com/content_ECCV_2018/html/Yihui_He_AMC_Automated_Model_ECCV_2018_paper.html.

[91] Y. Li and S. Ji, "L_0-ARM: Network sparsification via stochastic binary optimization", in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 2019, https://link.springer.com/chapter/10.1007/978-3-030-46147-8_26.

[92] J. Lin, C. Gan, and S. Han, "Defensive quantization: When efficiency meets robustness", in *International Conference on Learning Representations (ICLR)*, Workshop track, 2019, arXiv: 1904.08444.

[93] C. Liu, J. Zhuo, P. Cheng, R. Zhang, and J. Zhu, "Understanding and accelerating particle-based variational inference", PMLR, 2019, pp. 4082–4092, https://proceedings.mlr.press/v97/liu19i.

[94] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning", in *International Conference on Learning Representations (ICLR)*, 2019, arXiv: 1810.05270.

[95] C. Louizos, M. Reisser, T. Blankevoort, E. Gavves, and M. Welling, "Relaxed quantization for discretized neural networks", in *International Conference on Learning Representations (ICLR)*, 2019, arXiv: 1810.01875.

[96] Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. Dillon, B. Lakshminarayanan, and J. Snoek, "Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift", in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019, https://proceedings.neurips.cc/paper/2019/hash/8558cb408c1d76621371888657d2eb1d-Abstract.html.

[97] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library", in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019, https://arxiv.org/abs/1912.01703.

[98] D. Phan, N. Pradhan, and M. Jankowiak, "Composable effects for flexible and accelerated probabilistic programming in numpyro", *arXiv preprint arXiv:1912.11554*, 2019, https://arxiv.org/abs/1912.11554.

[99] A. S. Rekhi *et al.*, "Analog/mixed-signal hardware error modeling for deep learning inference", in *56th Annual Design Automation Conference*, ser. DAC, Association for Computing Machinery, 2019, DOI: 10.1145/3316781.3317770.

[100] W. Roth, G. Schindler, H. Fröning, and F. Pernkopf, "Training discrete-valued neural networks with sign activations using weight distributions", in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 2019, https://link.springer.com/chapter/10.1007/978-3-030-46147-8_23.

[101] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. Marculescu, "Single-path NAS: Designing hardware-efficient convnets in less than 4 hours", in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 2019, https://link.springer.com/chapter/10.1007/978-3-030-46147-8_29.

[102] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 2820–2828, DOI: https://10.1109/CVPR.2019.00293.

[103] Y. Umuroglu, D. Conficconi, L. Rasnayake, T. B. Preusser, and M. Själander, "Optimizing bit-serial matrix multiplication for reconfigurable computing", *ACM Transactions on Reconfigurable Technology and Systems*, vol. 12, no. 3, pp. 1–23, 2019, DOI: 10.1145/3369384.

[104] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-aware automated quantization with mixed precision", in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 8612–8620, https://openaccess.thecvf.com/content_CVPR_2019/html/Wang_HAQ_Hardware-Aware_Automated_Quantization_With_Mixed_Precision_CVPR_2019_paper.html.

[105] Z. Wang, T. Ren, J. Zhu, and B. Zhang, "Function space particle optimization for Bayesian neural networks", 2019, arXiv: 1902.09754.

[106] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 10 734–10 742, DOI: 10.1109/CVPR.2019.01098.

[107] M. Yin and M. Zhou, "ARM: augment-REINFORCE-merge gradient for stochastic binary networks", in *International Conference on Learning Representations (ICLR)*, 2019, arXiv: 1807.11143.

[108] C. Zhang, J. Bütepage, H. Kjellström, and S. Mandt, "Advances in variational inference", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 8, pp. 2008–2026, 2019, DOI: 10.1109/TPAMI.2018.2889774.

[109] Y. Zhang, R. Cai, A. Li, Y. Lin, Y. Chen, and H. H. Li, "Precision-aware training for deep neural networks on resistive crossbar systems", in *Proceedings of the Annual Design Automation Conference (DAC)*, 2019, pp. 1–6, DOI: 10.1145/3316781.3317924.

[110] J. Achterhold, J. M. Köhler, A. Schmeink, and T. Genewein, "Variational network quantization", in *International Conference on Learning Representations (ICLR)*, 2018, https://openreview.net/forum?id=ry-TW-WAb&trk=public_post_comment-text.

[111] A. G. Anderson and C. P. Berg, "The high-dimensional geometry of binary neural networks", in *International Conference on Learning Representations (ICLR)*, 2018, arXiv: 1705.07199.

[112] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, and S. Wanderman-Milne, "Jax: Composable transformations of python+numpy programs", *GitHub repository*, 2018, http://github.com/google/jax.

[113] R. Cai, A. Ren, N. Liu, C. Ding, L. Wang, X. Qian, M. Pedram, and Y. Wang, "Vibnn: Hardware acceleration of bayesian neural networks", *SIGPLAN Not.*, vol. 53, no. 2, pp. 476–488, Mar. 2018, DOI: 10.1145/3296957.3173212.

[114] C. Chen, R. Zhang, W. Wang, B. Li, and L. Chen, *A unified particle-optimization framework for scalable bayesian sampling*, 2018, arXiv: 1805.11659.

[115] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "TVM: An automated end-to-end optimizing compiler for deep learning", in *USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI, USA: USENIX Association, 2018, pp. 579–594, https://dada.cs.washington.edu/research/tr/2017/12/UW-CSE-17-12-01.pdf.

[116] S. Depeweg, J.-M. Hernandez-Lobato, F. Doshi-Velez, and S. Udluft, "Decomposition of Uncertainty in Bayesian Deep Learning for Efficient and Risk-sensitive Learning", en, in *International Conference on Machine Learning*, PMLR, Jul. 2018, pp. 1184–1193, https://proceedings.mlr.press/v80/depeweg18a.html.

[117] D. Duncan, A. Ó. hÉigeartaigh, and C. Ó. hÉigeartaigh, "On the effect of quantization on adversarial robustness", *arXiv preprint*, 2018, arXiv: 1809.07370.

[118] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices", in *Proceedings of the European Conference on Computer Vision (ECCV)*, Sep. 2018, https://openaccess.thecvf.com/content_ECCV_2018/html/Yihui_He_AMC_Automated_Model_ECCV_2018_paper.html.

[119] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks", in *European Conference on Computer Vision (ECCV)*, 2018, pp. 317–334, https://openaccess.thecvf.com/content_ECCV_2018/html/Zehao_Huang_Data-Driven_Sparse_Structure_ECCV_2018_paper.html.

[120] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference", in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2018, https://openaccess.thecvf.com/content_cvpr_2018/html/Jacob_Quantization_and_Training_CVPR_2018_paper.html.

[121] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference", in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 2704–2713, https://openaccess.thecvf.com/content_cvpr_2018/html/Jacob_Quantization_and_Training_CVPR_2018_paper.html.

[122] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper", *arXiv preprint*, 2018, arXiv: 1806.08342.

[123] X. Lin *et al.*, "All-optical machine learning using diffractive deep neural networks", *Science*, vol. 361, no. 6406, pp. 1004–1008, 2018, DOI: 10.1126/science.aat8084.

[124] X. Lin *et al.*, "All-optical machine learning using diffractive deep neural networks", *Science*, vol. 361, no. 6406, pp. 1004–1008, 2018, DOI: 10.1126/science.aat8084.

[125] X. Liu, Y. Ye, J. Tang, and X. Hu, "Robust neural network training via noise injection", in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2018, pp. 2596–2602, DOI: 10.24963/ijcai.2018/360.

[126] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K. Cheng, "Bi-Real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm", in *European Conference on Computer Vision (ECCV)*, 2018, pp. 747–763, DOI: 10.1007/978-3-030-01267-0_44.

[127] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through L_0 regularization", in *International Conference on Learning Representations (ICLR)*, 2018, arXiv: 1712.01312.

[128] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks", 2018, arXiv: 1802.05957, https://arxiv.org/abs/1802.05957.

[129] S. R. Nandakumar, M. Le Gallo, I. Boybat, B. Rajendran, A. Sebastian, and E. Eleftheriou, "A phase-change memory model for neuromorphic computing", *Journal of Applied Physics*, vol. 124, no. 15, p. 152 135, 2018, DOI: 10.1063/1.5042408.

[130] J. W. T. Peters and M. Welling, *Probabilistic binary neural networks*, 2018, arXiv: 1809.03368, https://arxiv.org/abs/1809.03368.

[131] M. Qin and D. Vucinic, *Noisy computations during inference: Harmful or helpful?*, 2018, arXiv: 1811.10649, https://arxiv.org/abs/1811.10649.

[132] J. Roesch, S. Lyubomirsky, L. Weber, J. Pollock, M. Kirisame, T. Chen, and Z. Tatlock, "Relay: A new ir for machine learning frameworks", in *SIGPLAN International Workshop on Machine Learning and Programming Languages*, ser. MAPL, New York, NY, USA: Association for Computing Machinery, 2018, pp. 58–68, DOI: 10.1145/3211346.3211348.

[133] G. Schindler, M. Zöhrer, F. Pernkopf, and H. Fröning, "Towards efficient forward propagation on resource-constrained systems", in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, M. Berlingerio, F. Bonchi, T. Gärtner, N. Hurley, and G. Ifrim, Eds., ser. Lecture Notes in Computer Science, vol. 11051, Springer, 2018, pp. 426–442, https://link.springer.com/chapter/10.1007/978-3-030-10925-7_26.

[134] O. Shayer, D. Levi, and E. Fetaya, "Learning discrete weights using the local reparameterization trick", in *International Conference on Learning Representations (ICLR)*, 2018, arXiv: 1710.07739.

[135] Y. Tsividis, "Not your father's analog computer", *IEEE Spectrum*, vol. 55, no. 2, pp. 38–43, 2018, DOI: 10.1109/MSPEC.2018.8278135.

[136] F. Tung and G. Mori, "Clip-q: Deep network compression learning by in-parallel pruning–quantization", in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, https://openaccess.thecvf.com/content_cvpr_2018/html/Tung_CLIP-Q_Deep_Network_CVPR_2018_paper.html.

[137] P. Warden, *Speech commands: A dataset for limited-vocabulary speech recognition*, 2018, arXiv: 1804.03209, https://arxiv.org/abs/1804.03209.

[138] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer, "Mixed precision quantization of convnets via differentiable neural architecture search", 2018, arXiv: 1812.00090, https://arxiv.org/abs/1812.00090.

[139] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks", in *International Conference on Learning Representations (ICLR)*, 2018.

[140] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "Netadapt: Platform-aware neural network adaptation for mobile applications", in *European Conference on Computer Vision (ECCV)*, Springer, 2018, pp. 285–300, DOI: 10.1007/978-3-030-01258-8_18.

[141] D. Zhang, J. Yang, D. Ye, and G. Hua, "LQ-Nets: Learned quantization for highly accurate and compact deep neural networks", in *European Conference on Computer Vision (ECCV)*, 2018, pp. 373–390, https://openaccess.thecvf.com/content_ECCV_2018/html/Dongqing_Zhang_LQ-Nets_Learned_Quantization_ECCV_2018_paper.html.

[142] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, *Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients*, 2018, arXiv: 1606.06160, https://arxiv.org/abs/1606.06160.

[143] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition", in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 8697–8710.

[144] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians", *Journal of the American Statistical Association*, vol. 112, no. 518, pp. 859–877, 2017, DOI: https://doi.org/10.1080/01621459.2017.1285773.

[145] Z. Cai, X. He, J. Sun, and N. Vasconcelos, "Deep learning with low precision by half-wave Gaussian quantization", in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5406–5414.

[146] J. V. Dillon, I. Langmore, D. Tran, E. Brevdo, S. Vasudevan, D. Moore, B. Patton, A. Alemi, M. Hoffman, and R. A. Saurous, "Tensorflow distributions", in *Proceedings of the International Conference on Learning Representations (ICLR) Workshop*, 2017, https://arxiv.org/abs/1711.10604.

[147] Y. Gal, R. Islam, and Z. Ghahramani, "Deep bayesian active learning with image data", in *Proceedings of the International Conference on Machine Learning (ICML)*, ser. PMLR, vol. 70, 2017, pp. 1183–1192, https://proceedings.mlr.press/v70/gal17a.html.

[148] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks", in *Proceedings of the International Conference on Machine Learning (ICML)*, 2017, pp. 1321–1330.

[149] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks", in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2017, pp. 1389–1397, DOI: 10.1109/ICCV.2017.154.

[150] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications", *arXiv preprint arXiv:1704.04861*, 2017, https://arxiv.org/abs/1704.04861.

[151] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017.

[152] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax", in *International Conference on Learning Representations (ICLR)*, 2017.

[153] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?", in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017, https://proceedings.neurips.cc/paper/2017/hash/2650d6089a6d640c5e85b2b88265dc2b-Abstract.html.

[154] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles", *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, pp. 6405–6416, 2017, https://proceedings.neurips.cc/paper_files/paper/2017/file/9ef2ed4b7fd2c810847ffa5fa85bce38-Paper.pdf.

[155] H. Li, S. De, Z. Xu, C. Studer, H. Samet, and T. Goldstein, "Training quantized nets: A deeper understanding", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 5811–5821.

[156] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets", in *International Conference on Learning Representations (ICLR)*, 2017, https://openreview.net/forum?id=SJGCiw5gl.

[157] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network", in *Neural Information Processing Systems (NeurIPS)*, 2017, pp. 345–353.

[158] Q. Liu, "Stein variational gradient descent as gradient flow", *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[159] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming", in *International Conference on Computer Vision (ICCV)*, 2017, pp. 2755–2763.

[160] C. Louizos, K. Ullrich, and M. Welling, "Bayesian compression for deep learning", in *Advances in Neural Information Processing Systems (NeurIPS)*, ser. NeurIPS, vol. 30, Curran Associates Inc., 2017, pp. 3290–3300, arXiv: 1705.08665.

[161] J. Luo, J. Wu, and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression", in *International Conference on Computer Vision (ICCV)*, 2017, pp. 5068–5076.

[162] D. Molchanov, A. Ashukha, and D. P. Vetrov, "Variational dropout sparsifies deep neural networks", in *International Conference on Machine Learning (ICML)*, 2017, pp. 2498–2507.

[163] Y. Shen *et al.*, "Deep learning with coherent nanophotonic circuits", *Nature Photonics*, vol. 11, no. 7, pp. 441–446, Jul. 2017, DOI: 10.1038/nphoton.2017.93.

[164] Y. Shen, N. C. Harris, S. Skirlo, M. Prabhu, T. Baehr-Jones, M. Hochberg, X. Sun, S. Zhao, H. Larochelle, D. Englund, and M. Soljačić, "Deep learning with coherent nanophotonic circuits", *Nature Photonics*, vol. 11, no. 7, pp. 441–446, 2017, DOI: 10.1038/nphoton.2017.93.

[165] TensorFlow Team, *Xla: Optimizing compiler for machine learning*, https://www.tensorflow.org/xla, Accessed 2025-09-26, 2017.

[166] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review", *IEEE Access*, vol. 5, pp. 17 322–17 341, 2017, DOI: 10.1109/ACCESS.2017.2742698.

[167] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. H. W. Leong, M. Jahre, and K. A. Vissers, "FINN: A framework for fast, scalable binarized neural network inference", in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (ISFPGA)*, 2017, pp. 65–74.

[168] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need", in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017, https://arxiv.org/abs/1706.03762.

[169] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms", 2017, arXiv: 1708.07747, https://arxiv.org/abs/1708.07747.

[170] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights", in *International Conference on Learning Representations (ICLR)*, 2017, arXiv: 1702.03044.

[171] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization", in *International Conference on Learning Representations (ICLR)*, 2017, arXiv: 1612.01064.

[172] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning", in *International Conference on Learning Representations (ICLR)*, 2017, arXiv: 1611.01578.

[173] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning", in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016, pp. 265–283, https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

[174] Y. Gal and Z. Ghahramani, *Bayesian convolutional neural networks with bernoulli approximate variational inference*, 2016, arXiv: 1506.02158.

[175] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning", in *International Conference on International Conference on Machine Learning*, ser. ICML, PMLR, 2016, pp. 1050–1059, https://proceedings.mlr.press/v48/gal16.html.

[176] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016, http://www.deeplearningbook.org.

[177] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNs", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016, pp. 1379–1387.

[178] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding", in *International Conference on Learning Representations ICLR*, Y. Bengio and Y. LeCun, Eds., 2016, arXiv: 1510.00149.

[179] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[180] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016, pp. 4107–4115.

[181] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, "Fixed point quantization of deep convolutional networks", in *International Conference on Machine Learning (ICML)*, 2016, pp. 2849–2858.

[182] Q. Liu and D. Wang, "Stein variational gradient descent: A general purpose Bayesian inference algorithm", *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 29, 2016.

[183] Z. Mariet and S. Sra, "Diversity networks: Neural network compression using determinantal point processes", in *International Conference on Learning Representations (ICLR)*, 2016.

[184] D. Miyashita, E. H. Lee, and B. Murmann, *Convolutional neural networks using logarithmic data representation*, 2016, arXiv: 1603.01025, https://arxiv.org/abs/1603.01025.

[185] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks", in *European Conference on Computer Vision (ECCV)*, 2016, pp. 525–542.

[186] W. Roth and F. Pernkopf, "Variational inference in neural networks using an approximate closed-form objective", in *NeurIPS Workshop on Bayesian Deep Learning*, 2016, http://bayesiandeeplearning.org/2016/papers/BDL_13.pdf.

[187] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016, pp. 2074–2082.

[188] S. Zagoruyko and N. Komodakis, "Wide residual networks", in *Proceedings of the British Machine Vision Conference (BMVC)*, 2016.

[189] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks", in *International Conference on Machine Learning (ICML)*, 2015, pp. 1613–1622, https://proceedings.mlr.press/v37/blundell15.html.

[190] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations", in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 28, Curran Associates, Inc., 2015, https://dl.acm.org/doi/10.5555/2969442.2969588.

[191] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples", in *International Conference on Learning Representations (ICLR)*, arXiv:1412.6572, 2015, https://arxiv.org/abs/1412.6572.

[192] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision", in *International Conference on Machine Learning (ICML)*, 2015, pp. 1737–1746.

[193] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2015, pp. 1135–1143.

[194] G. Hinton, O. Vinyals, and J. Dean, *Distilling the knowledge in a neural network*, 2015, arXiv: 1503.02531, https://arxiv.org/abs/1503.02531.

[195] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects", *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[196] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2015, pp. 2575–2583.

[197] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *International Conference on Learning Representations (ICLR)*, 2015, https://arxiv.org/abs/1412.6980.

[198] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", *Nature*, vol. 521, no. 7553, pp. 436–444, 2015, DOI: 10.1038/nature14539.

[199] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning", *arXiv preprint arXiv:1509.02971*, 2015, https://arxiv.org/abs/1509.02971.

[200] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural networks with few multiplications", in *International Conference on Learning Representations (ICLR)*, 2015.

[201] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows", in *Proceedings of the International Conference on Machine Learning (ICML)*, ser. PMLR, vol. 37, 2015, pp. 1530–1538, https://proceedings.mlr.press/v37/rezende15.html.

[202] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", in *International Conference on Learning Representations (ICLR)*, 2015.

[203] W. Sung, S. Shin, and K. Hwang, "Resiliency of deep neural networks under quantization", in *IEEE Workshop on Signal Processing Systems (SiPS)*, 2015, pp. 1–6, DOI: 10.1109/SiPS.2015.7345012.

[204] F. Abrol, S. Mandt, R. Ranganath, and D. Blei, "Deterministic annealing for stochastic variational inference", *stat*, vol. 1050, p. 7, 2014, https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=5cee21ae4605a330d9977164523be4b865df6ebd.

[205] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, "Deep speech: Scaling up end-to-end speech recognition", *arXiv preprint arXiv:1412.5567*, 2014, https://arxiv.org/abs/1412.5567.

[206] M. D. Hoffman, A. Gelman, *et al.*, "The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo.", *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1593–1623, 2014, https://www.jmlr.org/papers/volume15/hoffman14a/hoffman14a.pdf.

[207] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", 2014, arXiv: 1412.6980.

[208] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes", in *International Conference on Learning Representations (ICLR)*, arXiv: 1312.6114, 2014.

[209] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models", in *Proceedings of the International Conference on Machine Learning (ICML)*, ser. PMLR, vol. 32, 2014, pp. 1278–1286, https://proceedings.mlr.press/v32/rezende14.html.

[210] D. Soudry, I. Hubara, and R. Meir, "Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2014, pp. 963–971.

[211] Y. Bengio, N. Léonard, and A. Courville, *Estimating or propagating gradients through stochastic neurons for conditional computation*, 2013, arXiv: 1308.3432, https://arxiv.org/abs/1308.3432.

[212] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin, *Bayesian Data Analysis*, 3rd. Boca Raton, FL: Chapman & Hall / CRC, 2013.

[213] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference", *Journal of Machine Learning Research*, vol. 14, no. 1, pp. 1303–1347, May 2013, http://jmlr.org/papers/v14/hoffman13a.html.

[214] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 25, 2012, pp. 1097–1105, https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

[215] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 25, 2012.

[216] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012, https://dl.acm.org/doi/abs/10.5555/2380985.

[217] S. Brooks, A. Gelman, G. L. Jones, and X.-L. Meng, *Handbook of Markov Chain Monte Carlo*. CRC Press, 2011, DOI: https://doi.org/10.1201/b10905.

[218] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling", in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, ACM, 2011, pp. 365–376.

[219] A. Graves, "Practical variational inference for neural networks", in *Advances in Neural Information Processing Systems (NeurIPS)*, 2011, pp. 2348–2356.

[220] R. M. Neal, "Mcmc using hamiltonian dynamics", *Handbook of Markov Chain Monte Carlo*, vol. 2, no. 11, pp. 2–11, 2011, https://www.dam.brown.edu/people/geman/Homepage/CV/HandbookChapter5.pdf.

[221] J. Treibig, G. Hager, and G. Wellein, "LIKWID: Lightweight Performance Tools", in *2010 39th International Conference on Parallel Processing Workshops (ICPPW)*, IEEE, 2010, pp. 207–216, DOI: 10.1109/ICPPW.2010.38.

[222] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning", in *Proceedings of the International Conference on Machine Learning (ICML)*, ACM, 2009, pp. 41–48, DOI: 10.1145/1553374.1553380.

[223] A. D. Kiureghian and O. Ditlevsen, "Aleatory or epistemic? does it matter?", *Structural Safety*, vol. 31, no. 2, pp. 105–112, 2009, Risk Acceptance and Risk Communication, DOI: https://doi.org/10.1016/j.strusafe.2008.06.020.

[224] A. Krizhevsky, "Learning multiple layers of features from tiny images", University of Toronto, Tech. Rep., 2009.

[225] R. M. Zur, Y. Jiang, L. L. Pesce, and K. Drukker, "Noise injection for training artificial neural networks: A comparison with weight decay and early stopping", *Medical Physics*, vol. 36, no. 10, pp. 4810–4818, 2009, DOI: 10.1118/1.3213517.

[226] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda", *ACM Queue*, vol. 6, no. 2, pp. 40–53, 2008, DOI: 10.1145/1365490.1365500.

[227] P. D. Grünwald, *The minimum description length principle*. MIT press, 2007.

[228] T. Fawcett, "An introduction to roc analysis", *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006, DOI: https://doi.org/10.1016/j.patrec.2005.10.010.

[229] S. M. Pietralunga, L. Marazzi, and M. Martinelli, "Photon statistics of amplified spontaneous emission in dense wdm transmission systems", *IEEE Journal of Quantum Electronics*, vol. 39, no. 3, pp. 352–359, 2003, DOI: 10.1109/JQE.2003.809182.

[230] J. W. Goodman, *Statistical Optics*. Wiley, 2000.

[231] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, "An introduction to variational methods for graphical models", *Machine Learning*, vol. 37, no. 2, pp. 183–233, 1999, DOI: 10.1023/A:1007665907178.

[232] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998, DOI: 10.1109/5.726791.

[233] Y. Grandvalet and S. Canu, "Noise injection: Theoretical prospects", *Neural Computation*, vol. 9, no. 5, pp. 1093–1108, 1997, DOI: 10.1162/neco.1997.9.5.1093.

[234] R. M. Neal, *Bayesian Learning for Neural Networks*. Springer Science & Business Media, 1996, vol. 118, https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=db869fa192a3222ae4f2d766674a378e47013b1b.

[235] C. M. Bishop, "Training with noise is equivalent to tikhonov regularization", *Neural Computation*, vol. 7, no. 1, pp. 108–116, 1995, DOI: 10.1162/neco.1995.7.1.108.

[236] A. Murray and P. Edwards, "Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training", *IEEE Transactions on Neural Networks*, vol. 5, no. 5, pp. 792–802, 1994, DOI: 10.1109/72.317730.

[237] A. Gelman and D. B. Rubin, "Inference from iterative simulation using multiple sequences", *Statistical Science*, vol. 7, no. 4, pp. 457–472, 1992, DOI: 10.1214/ss/1177011136.

[238] C. J. Geyer, "Practical markov chain monte carlo", in *Statistical Science*, vol. 7, Institute of Mathematical Statistics, 1992, pp. 473–483, DOI: 10.1214/ss/1177011136.

[239] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon", in *Advances in Neural Information Processing Systems (NeurIPS)*, 1992, pp. 164–171.

[240] M. Höhfeld and S. E. Fahlman, "Learning with limited numerical precision using the cascade-correlation algorithm", *IEEE Transactions on Neural Networks*, vol. 3, no. 4, pp. 602–611, 1992.

[241] M. Höhfeld and S. E. Fahlman, "Probabilistic rounding in neural network learning with limited precision", *Neurocomputing*, vol. 4, no. 6, pp. 291–299, 1992.

[242] D. J. MacKay, "A practical Bayesian framework for backpropagation networks", *Neural computation*, vol. 4, no. 3, pp. 448–472, 1992.

[243] D. W. Scott, *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley, 1992.

[244] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage", in *Advances in Neural Information Processing Systems (NeurIPS)*, 1989, pp. 598–605.

[245] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, "Hybrid monte carlo", *Physics Letters B*, vol. 195, no. 2, pp. 216–222, 1987, DOI: 10.1016/0370-2693(87)91197-X.

[246] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", *Nature*, vol. 323, pp. 533–536, 1986, DOI: 10.1038/323533a0.

[247] G. Vannucci and M. C. Teich, "Computer simulation of superposed coherent and chaotic radiation", *Applied Optics*, vol. 19, no. 4, pp. 548–553, 1980, DOI: 10.1364/AO.19.000548.

[248] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications", *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970, DOI: 10.1093/biomet/57.1.97.

[249] K. Shimoda, H. Takahasi, and C. H. Townes, "Fluctuations in amplification of quanta with application to maser amplifiers", *Journal of the Physical Society of Japan*, vol. 12, no. 6, pp. 686–700, 1957, DOI: 10.1143/JPSJ.12.686.

[250] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines", *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953, DOI: 10.1063/1.1699114.

[251] S. Kullback and R. A. Leibler, "On information and sufficiency", *Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951, DOI: 10.1214/aoms/1177729694.

[252] H. Robbins and S. Monro, "A stochastic approximation method", *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951, DOI: 10.1214/aoms/1177729586.

[253] C. E. Shannon, "A mathematical theory of communication", *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948, DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[254] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion", *Physical Review*, vol. 36, no. 5, pp. 823–841, 1930, DOI: 10.1103/PhysRev.36.823.

[255] T. Bayes, "An essay towards solving a problem in the doctrine of chances", *Philosophical Transactions of the Royal Society of London*, vol. 53, pp. 370–418, 1763, DOI: 10.1098/rstl.1763.0053.