

Accelerated Execution of Bayesian Neural Networks using a Single Probabilistic Forward Pass and Code Generation

BERNHARD KLEIN, Heidelberg University, Germany

FALK SELKER, Heidelberg University, Germany

HENDRIK BORRAS, Heidelberg University, Germany

SOPHIE STEGER, Graz University of Technology, Austria

FRANZ PERNKOPF, Graz University of Technology, Austria

HOLGER FRÖNING, Heidelberg University, Germany

Machine learning models excel across various applications, such as medical diagnostic, weather forecasting, natural language processing and autonomous driving, yet their inadequate handling of uncertainty remains crucial for safety-critical applications. Traditional neural networks fail to recognize out-of-domain (OOD) data, often producing incorrect predictions without indicating uncertainty. Bayesian neural networks (BNNs) offer a principled framework for uncertainty estimation by providing statistically grounded probabilities alongside predictions.

Despite these advantages, BNNs suffer from high computational costs in both training and prediction. Each prediction requires sampling over weight distributions and executing multiple forward passes. To address this challenge, the Probabilistic Forward Pass (PFP) serves as an extreme approximation of Stochastic Variational Inference (SVI). While SVI assumes Gaussian-distributed weights without restrictions on activations, PFP extends this assumption to activations. This enables a fully analytical uncertainty propagation, replacing multiple forward passes with a single, more complex forward pass operating on probability distributions.

Thus, PFP requires specialized Gaussian-propagating operators absent from standard deep learning libraries. We present an end-to-end pipeline for training, compilation, optimization, and deployment of PFP-based BNNs on embedded ARM CPUs. By implementing custom operators in the deep learning compiler TVM, we develop a dedicated operator library for multilayer perceptrons and convolutional neural networks. Multiple operator implementations, along with manual and automatic tuning techniques, are applied to maximize efficiency. Ablation studies show that PFP consistently outperforms SVI in computational efficiency. For small mini-batch sizes, critical for low-latency applications, our approach achieves speedups of up to 4200 \times .

Our results show that PFP-based BNNs achieve performance comparable to SVI-BNNs on the Dirty-MNIST dataset in terms of classification accuracy, uncertainty quantification, and OOD detection, while significantly reducing computational overhead. These findings underscore the potential of combining Bayesian approximations with code generation and operator tuning to accelerate BNN predictions and enable their efficient deployment on resource-constrained embedded systems.

Additional Key Words and Phrases: Bayesian Neural Networks, Probabilistic Forward Pass, Code Generation, TVM, Lightweight Probabilistic Inference

1 INTRODUCTION

Machine learning (ML) based on deep (artificial) neural networks (DNNs) has revolutionized reasoning under uncertainty, and contributed to groundbreaking improvements in various prediction tasks including image, speech and signal processing. While it comes at tremendous cost, its excellent performance outweighs this downside, and various approaches are being pursued to bridge the gap between resource requirements and hardware capabilities. Specialized processor architectures are a notable example, which can substantially accelerate the execution of DNNs; however, they

Authors' addresses: Bernhard Klein, bernhard.klein@ziti.uni-heidelberg.de, Heidelberg University, Heidelberg, Germany; Falk Selker, falk.selker@neclab.eu, Heidelberg University, Heidelberg, Germany; Hendrik Borras, hendrik.borras@ziti.uni-heidelberg.de, Heidelberg University, Heidelberg, Germany; Sophie Steger, sophie.steger@tugraz.at, Graz University of Technology, Graz, Austria; Franz Pernkopf, pernkopf@tugraz.at, Graz University of Technology, Graz, Austria; Holger Fröning, holger.froening@ziti.uni-heidelberg.de, Heidelberg University, Heidelberg, Germany.

come at the cost of generality, often supporting only a limited set of operators, while at the same time, innovation in ML makes fast progress. As a result, the community has put a remarkable effort to the development of ML compilers, which map operators to special hardware primitives, for instance machine instructions for matrix-matrix multiply. This mapping involves several steps of considerable complexity, including loop and memory transformations and specifying which computations should be implemented with which machine instructions. Recent interest from the hardware and compilation community also extends to code generation for ML [13, 35]. This includes efforts in auto-tuning to identify operator implementations optimized for specific hardware architectures [33, 45, 51].

While DNNs are among the most effective methods for learning and predicting under uncertainty, they fundamentally lack the ability to quantify any uncertainty in a principled, mathematical manner. In classification tasks, for example, the softmax function is commonly used to produce class-wise output scores that are interpreted as probabilities. However, these softmax-derived values do not possess a rigorous probabilistic interpretation; they are best understood as confidence scores without a theoretical foundation grounded in probability theory. This limitation becomes particularly apparent when models are exposed to data samples that differ significantly from the training distribution. These are commonly referred to as *out-of-domain (OOD)* data. In such cases, neural networks often produce highly confident yet incorrect predictions, without any mechanism to signal elevated uncertainty or to indicate that the input lies outside the model’s domain of competence.

A theoretical framework to allow neural architectures to say "I don't know" is cast by Bayesian statistics, resulting in a class of probabilistic models called Bayesian neural networks (BNNs). They provide predictions in the form of a formal probability distribution, in which, in an simple setting, the mean would correspond to the prediction itself while the uncertainty is coded as variance or standard deviation of this distribution.

BNNs, while providing a mathematically rigorous framework for uncertainty estimation, incur an even greater computational burden than standard DNNs, which are already exceedingly expensive in both training and inference. Their high computational cost for predictions arises from the need to sample weights from posterior distributions and perform multiple forward passes to estimate predictive uncertainty. These repeated sampling and forward pass operations during prediction significantly increase latency, posing challenges for deploying BNNs in resource-constrained environments such as Internet of Things (IoT) devices and embedded systems.

In general, probabilistic modeling has gained significant interest within the ML community, leading to extensive related work [22, 42]. However, relatively little research addresses probabilistic modeling from a hardware perspective. Notable exceptions exist regarding hardware-efficient probabilistic circuits [37, 57]. Furthermore, a hardware-aware cost metric for probabilistic models has been introduced within tractable learning [21]. Specialized hardware accelerators for BNNs employing numerous on-chip pseudo-random number generators have also been proposed [9, 11, 54].

Probabilistic inference based on BNNs is a problem of exceptional complexity due to the inherent complexity of random variable algebra. At the same time, uncertainty is of high importance when bringing ML models into the "wild", where sensors are hindered by noise, occlusions, and previously unseen settings.¹ To successfully deploy BNNs on resource-constrained devices requires to combine multiple methods to bridge the gap among BNN requirements on memory and compute and the corresponding hardware capabilities.

One approach to mitigate this challenge is using partially Bayesian neural networks, treating only a subset of weights probabilistically [52]. This significantly reduces training complexity compared to full Bayesian models. Determining the appropriate number of probabilistic parameters remains an open question, as excessive reduction can degrade predictive

¹In San Francisco 2023 a driverless car just got stuck in wet concrete, since it was unaware of the situation that wet concrete exists. <https://www.nytimes.com/2023/08/17/us/driverless-car-accident-sf.html>

performance. Moreover, partial BNNs typically still require multiple forward passes for uncertainty estimation, limiting efficiency on resource-constrained devices. In contrast, last-layer methods avoid multiple forward passes by encoding probabilistic samples into multi-headed architectures [53]. Steger et al. demonstrates that a function-space-based repulsive loss effectively enables probabilistic inference even for pretrained models by only training an ensemble of probabilistic output layers [53].

A widely used approach to enable probabilistic reasoning in neural networks is Stochastic Variational Inference (SVI), which approximates the posterior distribution of weights by assuming a parameterized family of distributions [42], typically Gaussians. While SVI provides a tractable framework for Bayesian inference and is more efficient and GPU-friendly compared to Markov Chain Monte Carlo (MCMC) due to its reliance on gradient-based optimization and backpropagation, it still incurs substantial computational costs from weight sampling and the need for multiple forward passes [28]. An alternative solution is to adopt an analytical approximation, such as the Probabilistic Forward Pass (PFP) [47], which represents an extreme case of simplification by assuming not only Gaussian-distributed weights but also Gaussian-distributed activations. This approximation enables a closed-form solution for uncertainty estimation. While still treating all weights in a probabilistic manner, PFP requires only a single, albeit more complex, forward pass for both prediction and uncertainty estimation, eliminating the need for weight sampling and multiple forward passes.

Uncertainty in probabilistic machine learning is categorized as either *aleatoric* or *epistemic* [29]. Aleatoric uncertainty originates from inherent noise in the data and is irreducible, while epistemic uncertainty reflects model uncertainty due to limited knowledge and can be reduced with more data or better modeling. For classification problems, aleatoric uncertainty is typically quantified using Softmax Entropy (SME), while epistemic uncertainty—including out-of-distribution (OOD) samples—is measured using Mutual Information (MI). We refer to Section 2.3 for a detailed discussion.

Figure 1a illustrates the performance of a SVI-based BNN on images from three representative datasets. The training dataset MNIST [36] represents in-domain data, while Ambiguous-MNIST [41] introduces aleatoric uncertainty through samples between classes, and Fashion-MNIST [56] contains OOD images of fashion items. In the following, we will refer to these datasets in combination as "Dirty MNIST". Aleatoric uncertainty, quantified via SME, appears for ambiguous samples seen within one sample between classes; OOD images trigger confident but mutually disagreeing predictions. For each SVI sample, the raw unnormalized outputs, called logits, are typically normalized using a softmax function, and the most likely class is predicted by selecting the maximum value. Disagreement across predictions, quantified by MI, signals epistemic uncertainty, indicating potential OOD data.

Figure 1b demonstrates that reliable uncertainty quantification requires many samples, causing significant computational demands in traditional BNNs. For illustrative purposes, only three samples are shown in Figure 1a. The Figure also includes a Gaussian approximation that summarizes the logit samples using their mean and standard deviation. While this approach somewhat obscures detailed aleatoric and epistemic distinctions, it offers significant compactness benefits. PFP consistently utilizes this strategy through a single distribution-propagating forward pass. In practice, it effectively predicts and distinguishes both uncertainty types.

However, since PFP uses non-standard probabilistic operators, an efficient hardware-specific implementation of these operators is required before the method can be used in practice. These custom operators can be implemented using deep learning compilers, such as TVM [13], which enable automatic code generation and performance tuning [51]. By leveraging these tools, the rather complex operations of the Probabilistic Forward Pass can be optimized for specific hardware architectures, allowing BNNs to run more efficiently in resource-limited environments while maintaining robust uncertainty estimation.

In detail, this work makes the following contributions:

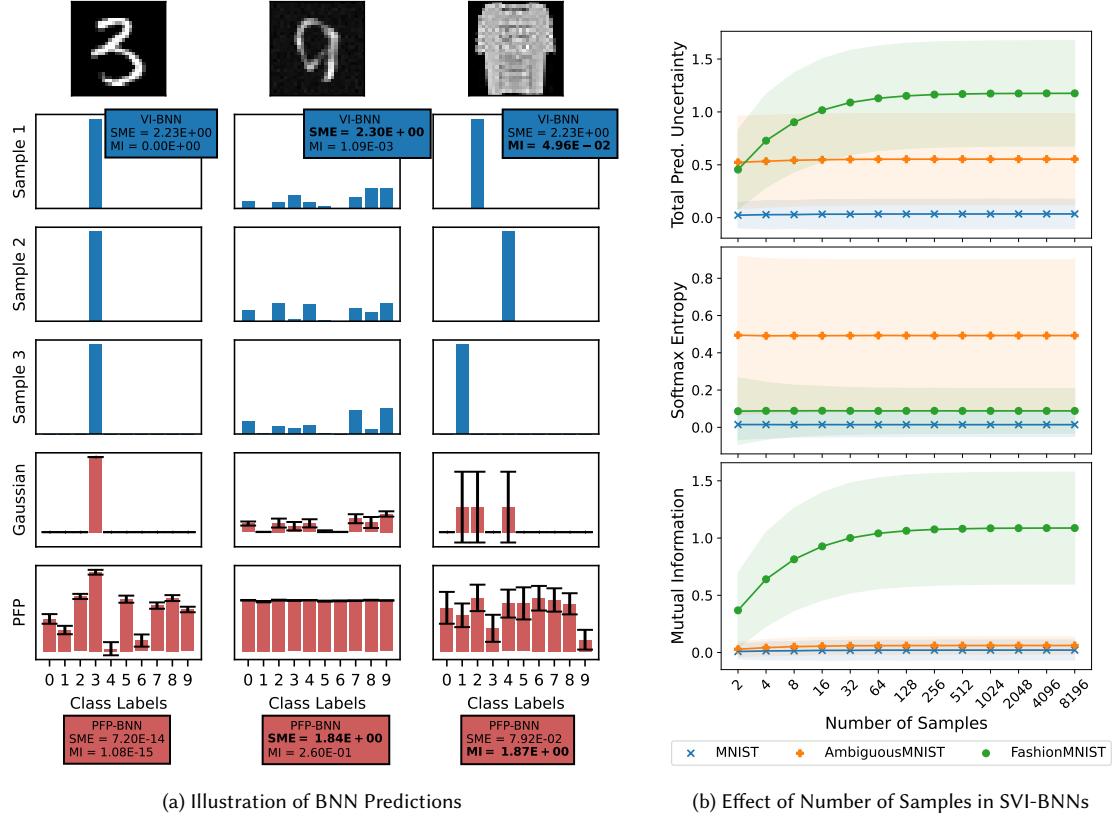


Fig. 1. (a) Exemplary predictions from a SVI-BNN (blue), its Gaussian representation, and PFP [47], showcasing MNIST [36] and Ambiguous-MNIST [41] as in-domain examples, and Fashion-MNIST [56] as an out-of-domain sample. Variability in class predictions demonstrates aleatoric uncertainty (higher Softmax Entropy, SME), whereas variability across predictive samples indicates epistemic uncertainty (higher Mutual Information, MI). SVI and PFP effectively quantify these uncertainties. Note: Only three SVI samples shown, insufficient for robust estimation. (b) Influence of predictive sample count on uncertainty metrics. Softmax Entropy (aleatoric uncertainty) remains stable, while Total Predictive Uncertainty and Mutual Information (epistemic uncertainty), especially for out-of-domain data (Fashion-MNIST), require more samples for reliable OOD detection.

- We propose a training pipeline based on Stochastic Variational Inference and demonstrate that the Probabilistic Forward Pass achieves comparable performance in uncertainty estimation and out-of-domain detection, validated on the "Dirty MNIST" dataset.
- We extend the deep learning compiler TVM by integrating essential probabilistic operators for multi-layer perceptron (MLP) and convolutional neural network (CNN) architectures, enabling the efficient execution of PFP-based models on ARM processors.
- We implement optimized execution schedules as well as manual and automatic tuning strategies for computationally intensive PFP operators, ensuring high performance on resource-constrained ARM CPUs.
- We evaluate the practical feasibility of our approach through benchmarks on embedded ARM processors, demonstrating significant performance improvements that facilitate the deployment of BNNs for uncertainty-aware edge and IoT applications.

As we will see, although the PFP concept relies on highly non-standard operators, casting the problem within a mapping framework such as TVM enables deployment without specialized libraries while also allowing performance optimizations with reasonable effort. In more detail, we address the following research questions and aim to provide corresponding answers in what follows:

- (1) How good is the quality of PFP-based Bayesian neural architectures?
- (2) How well does TVM support custom operator implementation?
- (3) How fast is the PFP approach, when employing various optimization techniques, compared to a SVI-based BNN?

The remainder of this paper is structured as follows. Section 2 reviews related work on efficient BNN implementations and their deployment on resource-constrained devices. Section 3 presents the theoretical foundation and Section 4 training, and uncertainty evaluation of PFP. Section 5 discusses the implementation of custom operators using TVM, highlighting the specific challenges associated with PFP operators. Section 6 covers operator tuning, profiling, and benchmarking, comparing PFP- and SVI-based BNNs on embedded ARM processors. Finally, Section 7 summarizes the key findings and conclusions of this work. The presented PFP operator library is publicly available.²

2 BACKGROUND AND RELATED WORK

This section provides a brief introduction to BNN methods, with a particular focus on related work concerning efficient BNN inference. We further clarify the distinction between different types of uncertainty and review methods for efficient inference on resource-constrained embedded systems, specifically in the context of BNNs.

2.1 Bayesian Neural Networks (BNNs)

Bayesian neural networks integrate Bayesian inference (BI) into neural networks by treating weights as random variables rather than fixed values [43]. Weights are sampled from distributions, enabling uncertainty estimation and improved generalization, particularly in data-scarce or noisy settings. BNNs place a prior distribution over the network’s weights, and then, given observed data, compute a posterior distribution. The prior reflects any previous knowledge about the weight values, while the posterior captures the updated beliefs after seeing the data [28, 42]. This probabilistic framework quantifies predictive uncertainty, making BNNs valuable for uncertainty-sensitive and safety-critical tasks.

In BNNs, weight distributions can take various forms, but Gaussian approximations are common due to their mathematical properties and computational efficiency. Computing the posterior distribution of weights is in general analytically intractable due to the high-dimensional and non-convex nature of neural networks [28]. Training and prediction involve sampling weights from these distributions, unlike deterministic networks that use fixed point estimates. This sampling enables BNNs to produce a distribution of possible outcomes rather than a single prediction. Such distributions facilitate uncertainty estimation and can help detect out-of-domain data.

The equivalent of training in this context is performing Bayesian inference to learn this weight distribution [28]. BI techniques are commonly categorized into two classes: sampling-based methods and variational methods that optimize an approximate posterior distribution. Markov Chain Monte Carlo (MCMC) methods, for instance, sample directly from the posterior distribution, making them particularly well-suited for BNNs [8]. These methods construct a Markov chain with the target posterior as its equilibrium distribution. Hamiltonian Monte Carlo (HMC) [44] in combination with the No-U-Turn-Sampler (NUTS) [26] is considered the gold standard in terms of quality and reliability. Although

²The code is available at: <https://github.com/UniHD-CEG/PFP-Operator-Library>

offering strong theoretical guarantees, MCMC methods suffer from high computational costs due to slow convergence in high-dimensional spaces and the requirement for a large number of samples.

Variational Inference (VI) reformulates BI as an optimization problem, improving scalability for large datasets and models [5, 25]. It introduces a variational distribution $q_\phi(\theta)$ to approximate the posterior $p(\theta | \mathcal{D})$ and minimizes the Kullback-Leibler (KL) divergence $\text{KL}(q_\phi(\theta) \parallel p(\theta | \mathcal{D}))$. The optimization maximizes the Evidence Lower Bound (ELBO):

$$\text{ELBO} = \mathbb{E}_{q_\phi(\theta)} [\log p(\mathcal{D} | \theta)] - \text{KL}(q_\phi(\theta) \parallel p(\theta | \mathcal{D})),$$

balancing data fit and prior regularization [31]. In this work, we use the terms Variational Inference and Stochastic Variational Inference (SVI) interchangeably to refer to the general concept of variational approximation; however, all experiments are conducted using stochastic optimization, thus SVI. SVI leverages mini-batches and gradient-based optimization, enabling efficient training on GPUs and scalability to deep neural networks.

While MCMC offers a theoretically sound approach for posterior estimation but is computationally expensive and difficult to scale, SVI offers a more practical alternative, trading off accuracy for improved scalability and faster convergence, making it a popular choice for BNNs in modern deep learning applications.

Bayesian Neural Networks and Efficient Inference. Over the years, a wide range of methodologies for BNNs has been developed; for a comprehensive survey, we refer to [28, 42]. In the following, we highlight some of the most pioneering works that focus on approximate inference techniques designed to enable efficient prediction with BNNs. While SVI and MCMC methods are mathematically grounded many alternative approaches only loosely relate to these foundational principles. This often enables improved computational efficiency, but comes at the cost of reduced theoretical rigor and degraded uncertainty estimation performance.

To reduce computational cost, Blundell et al. proposed an SVI-based approach that approximates the posterior using a simplified distribution [6]. This approach reduces computational costs compared to MCMC but remains challenging for deployment on resource-constrained platforms such as mobile devices. Gal and Ghahramani advanced BNN efficiency by proposing Monte Carlo Dropout [19, 20], which approximates BNNs by applying dropout at inference. This method provides uncertainty estimates with minimal computational overhead, interpreting dropout as a probabilistic mechanism, making it well-suited for constrained environments. However, its approximation quality varies notably.

Deep Ensembles (DE) are another prominent example in this category of algorithms. They provide uncertainty estimates by training multiple independent neural networks and aggregating their predictions [34]. Due to their simplicity and effectiveness they are widely used. A fundamental limitation shared with MCMC methods is the substantial memory overhead at inference time, as both approaches require maintaining multiple sets of model parameters or posterior samples, making them impractical for deployment on resource-constrained embedded devices. Repulsive ensembles [15] extend Deep Ensembles by promoting diversity among ensemble members. Steger et al. [53] show that a function-space repulsive loss enables probabilistic inference even for pretrained models by training only an ensemble of probabilistic output layers.

Overall MCDO and ensemble methods have all improved efficient Bayesian inference significantly. However, as a common theme they all discard the formal theoretical foundations, found in SVI and MCMC methods. While these foundational works have advanced efficient Bayesian inference, direct applications to mobile processors remain limited.

2.2 Types of Uncertainty

In the probabilistic ML context uncertainty is categorized into *aleatoric and epistemic uncertainty* [29]. *Aleatoric uncertainty* stems from inherent noise in the data, making it irreducible even with unlimited data. It arises when input features contain measurement noise or the output variable is intrinsically unpredictable. Aleatoric uncertainty can be *homoscedastic* (constant across inputs) or *heteroscedastic* (input-dependent).

Epistemic uncertainty arises from a lack of knowledge, such as missing data or suboptimal models. It is reducible and decreases with more data or model refinement. This uncertainty is prominent in underrepresented input regions, including OOD data. The key distinction is that aleatoric uncertainty is data-driven and irreducible, while epistemic uncertainty is model-related and can be reduced.

BNNs distinguish between aleatoric and epistemic uncertainty in both regression and classification tasks. In regression, aleatoric uncertainty is modeled via a heteroscedastic head that predicts data-dependent noise. For classification, aleatoric and epistemic uncertainty are commonly estimated via Softmax Entropy and Mutual Information, respectively [41].

Both MCMC and SVI require sampling from the posterior weight distribution. If N predictive samples are needed, N complete weight sets must be drawn, and N forward passes executed. As shown in Figure 1a, each predictive sample in classification yields distinct predicted logits. For classification, total predictive uncertainty is quantified using *Shannon Entropy* [50]:

$$\mathbb{H}(y|x, \mathcal{D}) = - \sum_c^K \left[\frac{1}{N} \sum_n^N (p(y=c|x, w_n)) \cdot \log \left(\frac{1}{N} \sum_n^N (p(y=c|x, w_n)) \right) \right] \quad (1)$$

Here $p(y=c|x, w_n)$ denotes the probability of prediction y and ground-truth class c being identical, given input x and weight sample w_n from the n -th sample. \mathcal{D} denotes the full dataset, K the number of classes, and N the number of samples. This total predictive uncertainty is decomposed into their aleatoric and epistemic components, measured by the *Softmax Entropy*,

$$\mathbb{E}_{p(w|\mathcal{D})}(\mathbb{H}(y|x, w)) = -\frac{1}{N} \sum_n^N \sum_c^K p(y=c|x, w_n) \cdot \log(p(y=c|x, w_n)) \quad (2)$$

and *Mutual Information* [14]

$$\mathbb{I}(y, w|x, \mathcal{D}) = \mathbb{H}(y|x, \mathcal{D}) - \mathbb{E}_{p(w|\mathcal{D})}(\mathbb{H}(y|x, w)). \quad (3)$$

For the Shannon Entropy the samples are summed first, while the Softmax Entropy sums first over classes K , and thereby encodes the mean of the aleatoric uncertainty.

2.3 Efficient Inference of Neural Architectures on Resource-constrained Devices

Resource-efficient uncertainty estimation in neural networks, particularly BNNs, has become an increasingly important area of research, especially given the growing demand for deploying sophisticated machine learning models on resource-constrained devices such as mobile processors.

Resource-Efficient Deployment on Mobile Devices. Deploying machine learning models on mobile processors requires careful optimization for computational and memory constraints. Chen et al. introduced TVM [13], an automated deep learning compiler that optimizes models for diverse hardware backends, including mobile ARM processors. TVM facilitates efficient deployment of deep neural architectures by automating optimizations and supporting hardware-specific enhancements such as quantization and model pruning.

Quantization [27] reduces the precision of neural network weights and activations, significantly lowering inference resource requirements. Deep compression methods, including pruning and quantization [23, 49], reduce the resource footprint, enhancing applicability to mobile processors. Neural architecture search and automatic compression, combined with reinforcement learning [10, 12, 24, 32], automate architecture refinement and compression parameter selection, optimizing bit width and sparsity, and enabling compression at the layer, channel, or block level. For a more detailed overview of resource-efficient neural network inference on embedded systems, including a comparison of various compression methods and hardware architectures, we refer to [48].

Considering the intersection of Bayesian methods and neural networks from a different perspective, Louizos et al. propose Bayesian compression techniques for DNNs [40], leveraging priors to guide pruning while using posterior uncertainties to determine optimal fixed-point precision. Although relevant for mobile DNN deployments, this approach does not address the efficiency of BNNs.

Bayesian Neural Networks on Resource-Constrained Devices. Although direct research on deploying BNNs on mobile processors is limited, several works address related areas. A closely related effort is by Banerjee et al., who propose AcMC2, a compiler that transforms probabilistic models into optimized MCMC-based hardware accelerators, such as FPGAs or ASICs [3]. However, this work targets general probabilistic models, which differ significantly in scale and underlying principles from BNNs.

For specialized accelerators based on standard CMOS technology, ShiftBNN [54], B^2N^2 [2] and VIBNN [11] leverage ASICs and FPGAs, respectively, to accelerate BNN training and inference using large-scale Gaussian random number generators. Beyond CMOS, early research explores alternative technologies, such as analog computing and resistive memory, to introduce stochasticity as a source of uncertainty in Bayesian methods. Examples include BNNs on probabilistic photonic hardware [9], custom hardware for probabilistic circuits [57], and Bayesian networks utilizing resistive memory and its inherent operational uncertainty [7, 38, 39]. However, these studies remain in early stages with limited experimental validation, leaving the feasibility of these technologies uncertain.

In summary, the intersection of BNNs and mobile processors remains an emerging area of research. While foundational methods such as MCDO and SVI offer pathways toward efficient uncertainty estimation, the challenge of deploying these methods on resource-constrained devices like mobile ARM processors is still largely unexplored. Tools like TVM, combined with techniques such as quantization and model partitioning, provide promising avenues for bringing the benefits of Bayesian inference to mobile platforms. Our work aims to build on these foundations by mapping efficient Stochastic Variational Inference methods to mobile ARM processors using TVM, thereby addressing the unique challenges posed by this constrained environment.

3 PROBABILISTIC FORWARD PASS

The Probabilistic Forward Pass [46, 47] efficiently implements BNNs by propagating probability distributions through the network, eliminating the need for repeated sampling and multiple forward passes. This approach significantly reduces computational overhead while preserving predictive uncertainty estimation.

PFP represents an extreme approximation of Stochastic Variational Inference-based BNNs (SVI-BNNs). While SVI-BNNs typically assume Gaussian-distributed weights with a mean-field independence assumption, PFP extends this to Gaussian-distributed activations. This assumption leverages the central limit theorem, which states that the sum of many independent random variables tends toward a normal distribution. Unlike general SVI, which can model complex activation distributions, PFP constrains activations to a Gaussian form.

This restriction enables a single closed-form forward pass, eliminating the need for stochastic sampling and multiple evaluations. As a result, PFP is particularly suited for resource-constrained environments, where computational efficiency and reliable uncertainty estimation are critical. Input distributions are transformed into output distributions through linear and non-linear operations adapted to handle Gaussian distributions.

Consider a fully connected layer where the input follows a Gaussian distribution with mean and variance $(\mu_{in}, \sigma_{in}^2)$, and the weights are Gaussian with parameters (μ_w, σ_w^2) . The output pre-activation distribution is derived by computing the mean and variance of the linear transformation. The mean is obtained by propagating expectations, while the variance incorporates uncertainties from both weights and inputs. PFP considers activations to be independent, ensuring variance computations remain tractable. Equations 4 and 5 illustrate the scalar computation of mean and variance for the l -th fully connected layer,

$$\mu_{a_i^l} = \sum_{j=1}^{d_{l-1}} \mu_{w_{ij}^l} \cdot \mu_{x_j^{l-1}} \quad (4)$$

$$\sigma_{a_i^l}^2 = \sum_{j=1}^{d_{l-1}} \sigma_{w_{ij}^l}^2 \cdot \mathbb{E} \left[\left(x_j^{l-1} \right)^2 \right] + \mu_{w_{ij}^l}^2 \cdot \left(\mathbb{E} \left[\left(x_j^{l-1} \right)^2 \right] - \mu_{x_j^{l-1}}^2 \right). \quad (5)$$

Thereby the *second raw moment* $\mathbb{E}(x^2) = \mu^2 + \sigma^2$ is used. Moreover, d_{l-1} denotes the width of the previous layer, which is the input activation tensor width of the current layer. Reformulation to use means and variances instead of *second raw moments* is possible,

$$\sigma_a^2 = \sigma_w^2 \cdot \mathbb{E}[x^2] + \mu_w^2 \cdot (\mathbb{E}[x^2] - \mu_x^2) \quad (6)$$

$$= \sigma_w^2 \cdot \mu_x^2 + \mu_w^2 \cdot \sigma_x^2 + \sigma_w^2 \cdot \sigma_x^2 \quad (7)$$

illustrated here in tensor notation. Later implementations support both variants, selecting the one with lower computational overhead. To avoid unnecessary conversions, the output of one layer and the input of the next must be consistently represented, either as mean and variance or as mean and *second raw moment* tuples.

A key challenge in propagating distributions arises from non-linear activation functions. For the widely used Rectified Linear Unit (ReLU), PFP approximates post-activation outputs by matching the first two moments (mean and variance) of the original output distribution. Figure 2 illustrates this moment-matching process, where an input truncated Gaussian is transformed back into a proper Gaussian distribution. Thus, the PFP-specific ReLU operator, while still an element-wise

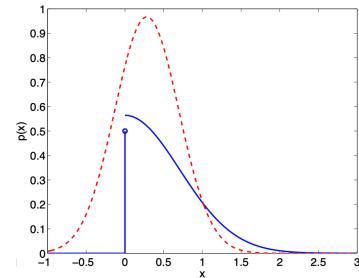


Fig. 2. Illustration of Gaussian moment matching for ReLU activations. The true distribution (solid line) is approximated as a Gaussian (dashed line). Reproduced with permission from [47].

operation, is more complex than its deterministic counterpart, $\text{ReLU}(x) = \max(0, x)$,

$$\mu_{x_i^l} = \mathbb{E}[x_i^l] = \frac{\mu_{a_i^l}}{2} \left(1 + \operatorname{erf} \left(\frac{\mu_{a_i^l}}{\sqrt{2\sigma_{a_i^l}^2}} \right) \right) + \sqrt{\frac{\sigma_{a_i^l}^2}{2\pi}} \exp \left(-\frac{\mu_{a_i^l}^2}{2\sigma_{a_i^l}^2} \right) \quad (8)$$

$$\mathbb{E}[(x_i^l)^2] = \frac{\sigma_{a_i^l}^2 + \mu_{a_i^l}^2}{2} \left(1 + \operatorname{erf} \left(\frac{\mu_{a_i^l}}{\sqrt{2\sigma_{a_i^l}^2}} \right) \right) + \mu_{a_i^l} \sqrt{\frac{\sigma_{a_i^l}^2}{2\pi}} \exp \left(-\frac{\mu_{a_i^l}^2}{2\sigma_{a_i^l}^2} \right) \quad (9)$$

where erf denotes the error function $\operatorname{erf}(u) = \frac{2}{\sqrt{\pi}} \int_0^u \exp(-z^2) dz$ [46]. As distributions propagate through successive layers, the output layer generates the final predictive distribution.

A key advantage of PFP is its ability to compute the expected log-likelihood in closed form, eliminating the need for sampling. At test time, predictions require only a single forward pass, directly computing expected outputs and uncertainties without costly sampling or model averaging. By reformulating neural network computations to operate on distributions, PFP offers a scalable and efficient approach for deploying BNNs in practical settings.

3.1 Conceptual Limitations

As discussed in Section 2.3, classification tasks typically quantify total uncertainty using Shannon Entropy, while Softmax Entropy and Mutual Information estimate aleatoric and epistemic uncertainty, respectively. PFP, with its single-forward-pass computation, predicts the means and variances of the logits, eliminating the need for multiple stochastic samples. However, this approach inherently assumes Gaussian distributions for the logits, which imposes a limitation. If the true predictive distribution exhibits high variability or deviates significantly from a Gaussian distribution, the approximation becomes inaccurate.

While Shannon Entropy, as an average over samples, remains largely unaffected due to its dependence on mean predictions, Softmax Entropy is more sensitive. Since Softmax Entropy relies on the full distribution of sampled predictions, inaccuracies in the Gaussian approximation can propagate, affecting Mutual Information estimation.

In an artificially constructed high epistemic uncertainty scenario, generated by assigning random one-hot encoded class predictions, the total uncertainty remains identical between the Gaussian approximation and the sample-based estimate reflecting the true distribution. However, the Gaussian approximation underestimates Mutual Information by 44 %, illustrating a substantial deviation in epistemic uncertainty quantification. While this constitutes an extreme case, it fundamentally highlights the limitation of the Gaussian assumption in accurately disentangling epistemic and aleatoric uncertainty when the underlying predictive distributions deviate substantially from a Gaussian.

4 PFP TRAINING AND UNCERTAINTY ESTIMATION EXPERIMENTS

A key advantage of PFP is its compatibility with pretrained SVI models. It benefits from SVI’s relatively fast training while leveraging established tools for creating SVI-based BNNs. Probabilistic Programming Languages (PPLs) such as Pyro [4], Stan³, and TensorFlow Probability⁴ excel in designing, training, and inferring probabilistic models. In this work, BNNs are trained using Pyro SVI and exported for use with PFP.

³<https://mc-stan.org/>

⁴<https://github.com/tensorflow/probability>

Two neural architectures were used in the experiments: a simple multi-Layer perceptron (MLP) with one hidden layer of 100 neurons and a LeNet-5 [36] architecture. In both cases, all weights are treated probabilistically. Gaussian priors are used for the learnable parameters. Additionally, the mean-field assumption [16] simplifies learning by neglecting correlations between Gaussian weight distributions.

SVI, as a BNN training method, closely resembles standard neural network training using gradient descent with a specialized loss terms. However, its complexity exceeds that of non-probabilistic training. Due to the multi-objective nature of the optimization, training time tends to be longer, and selecting appropriate hyperparameters and initial values is crucial. The SVI-BNNs are trained for 1000 epochs using Adam [30] with a constant learning rate of 0.001. The variational posterior weights are initialized with $\mu = 0.08$ and $\sigma = 0.0001$, and a mini-batch size of 100 is used.

Balancing the expected log-likelihood term and the KL-divergence term in the ELBO using a constant factor α is challenging. However, dynamically increasing the KL term over epochs, known as *KL annealing* [1, 58], has proven more effective and robust. The dynamically adapted ELBO,

$$\text{ELBO}(q, e) = \mathbb{E}[\log p(\mathcal{D}|w)] - A(e) \cdot \text{KL}(q(w)||p(w|\mathcal{D})), \quad (10)$$

uses a linearly increasing KL factor $A(e)$ that scales from 0 to $\alpha_{\max} = 0.25$ over the training epochs e . KL annealing mitigates sensitivity to initialization values and removes the need for non-probabilistic pretraining.

The trained means and variances of each weight can be directly utilized by PFP, requiring only a conversion from logarithmic to normal representation, followed by an uncertainty calibration—a common procedure when transferring distributions between probabilistic methods. This calibration involves a global reweighting of the variances. We refer to the heuristically determined scaling parameter as the calibration factor.

To evaluate the BNNs' ability to report both high and low uncertainties while distinguishing between aleatoric and epistemic uncertainty, we use the dataset introduced by Mukhoti et al., which extends MNIST with Ambiguous-MNIST for aleatoric uncertainty and Fashion-MNIST for epistemic uncertainty. We refer to this combined dataset as Dirty-MNIST. Following their argument that aleatoric uncertainty is unavoidable in real-world scenarios, we train on a combined dataset of MNIST and Ambiguous-MNIST, while the OOD dataset Fashion-MNIST remains unseen [41].

4.1 Qualitative Performance

To assess the effectiveness of PFP-based BNNs as an approximation to sampling-based SVI-BNNs, we compare their predictions on the same datasets. Figure 1a illustrates the performance on individual inputs, while Figure 3 compares key uncertainty metrics: Shannon Entropy for total uncertainty, Softmax Entropy for aleatoric uncertainty, and Mutual Information for epistemic uncertainty. These metrics aggregate over the sample dimension, and as shown in Figure 1b, the number of samples significantly impacts Shannon Entropy, thereby influencing Mutual Information. Unlike SVI-BNNs, PFP does not inherently provide a sample dimension. To ensure a fair comparison, we introduce an artificial sampling dimension using PFP-predicted means (μ_{PFP}) and variances (σ_{PFP}^2) of the logits. Assuming a Gaussian distribution, logit samples l_{PFP} are generated as:

$$l_{\text{PFP}} \sim \mathcal{N}(\mu_{\text{PFP}}, \sigma_{\text{PFP}}^2). \quad (11)$$

This *logit sampling* approach is computationally efficient, as it avoids sampling within the network and eliminates the need for multiple forward passes. It serves as a post-processing step, enabling standard uncertainty metrics typically used for sampling-based BNNs. In resource-constrained applications, directly leveraging PFP-predicted variances (σ_{PFP}^2) for decision-making is an efficient alternative.

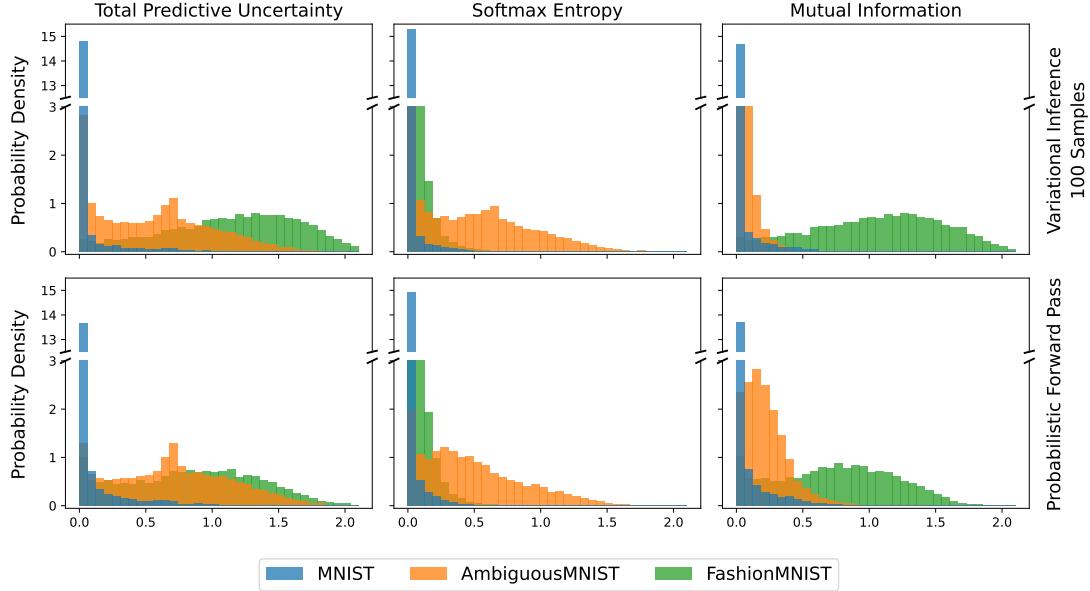


Fig. 3. Comparison of SVI and PFP uncertainty predictions. For MNIST, both uncertainties are expected to be low; Ambiguous-MNIST exhibits higher aleatoric uncertainty (Softmax Entropy), and Fashion-MNIST, as OOD data, shows higher epistemic uncertainty (Mutual Information). Both methods effectively assign the majority of images to their respective domains.

Table 1. Performance Comparison of SVI and PFP-based BNNs.

Architecture	Stochastic Variational Inference		Probabilistic Forward Pass		
	Accuracy (%)	AUROC	Calibration Factor	Accuracy (%)	AUROC
MLP	96.3	0.812	0.3	96.3	0.858
LeNet-5	98.7	0.986	0.4	98.9	0.966

Figure 3 shows that both methods yield higher total predictive uncertainty for Ambiguous-MNIST and Fashion-MNIST compared to the lower uncertainties observed for standard MNIST. Similarly, both exhibit elevated Softmax Entropy for Ambiguous-MNIST, indicating increased aleatoric uncertainty, and higher Mutual Information for the out-of-distribution Fashion-MNIST dataset, reflecting greater epistemic uncertainty. Overall, both approaches provide uncertainty estimates consistent with expectations and desired behavior.

A detailed analysis of Softmax Entropy and Mutual Information across all images, shown in Figure 4, indicates that SVI outperforms PFP in disentangling aleatoric and epistemic uncertainty, as theoretically expected. However, from a practical standpoint, PFP still provides sufficient separation in most cases. Only in certain edge cases do both uncertainties exhibit high values, resulting in less distinct separation.

The Area Under the Receiver Operating Characteristic Curve (AUROC) quantifies a model's ability to distinguish between in-domain and out-of-domain samples. It is defined as the area under the ROC curve, which plots the true positive rate (TPR) against the false positive rate (FPR) across various thresholds, i.e., $\text{AUROC} = \int_0^1 \text{TPR}(\text{FPR}^{-1}(x)) dx$. Thereby a AUROC of 1.0 indicates perfect discrimination, while a score of 0.5 corresponds to random guessing. For a detailed discussion on AUROC computation, refer to [17].

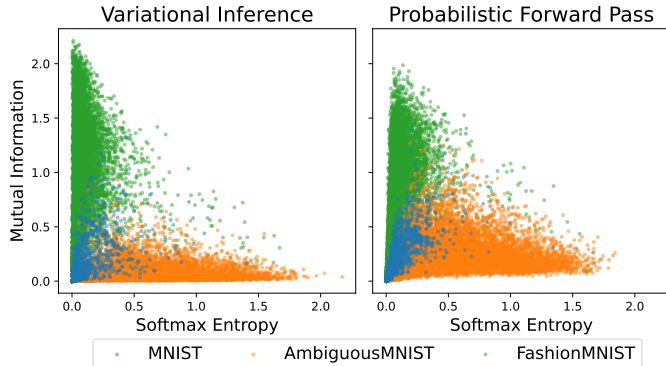


Fig. 4. Disentangled Epistemic (MI) and Aleatoric (SME) Uncertainty. Comparison of SVI and PFP uncertainty predictions. While PFP performs comparably to SVI in estimating total uncertainty, its ability to disentangle aleatoric and epistemic uncertainties is somewhat limited, as anticipated. Nevertheless, the practical distinction remains sufficiently robust for most cases.

Table 1 presents a comparative analysis of SVI and PFP using AUROC as a metric to evaluate their effectiveness in OOD detection. The results indicate that both methods exhibit comparable performance in terms of predictive accuracy and OOD detection capability.

Furthermore, the influence of the neural architecture is evident, as both methods demonstrate improved predictive performance and OOD detection when utilizing a more expressive convolutional neural network. In summary, PFP achieves prediction quality closely aligned with SVI in both accuracy and OOD detection.

5 EXTENDING TVM WITH A PFP OPERATOR LIBRARY

TVM provides multiple internal languages and intermediate representations, including TensorIR [18], TensorExpression (TE), TVMScript, and Relax [33], which are essential for implementing custom operators. While TE defines computational rules in a very succinct way, TensorIR defines the interaction of computations more fine-grained as modular *blocks*, enabling flexible scheduling and optimizations.⁵ Relax [33], the successor to Relay [45], serves as a high-level intermediate representation, supporting dynamic shapes, control flow, and seamless integration with TensorIR.⁶ TVMScript, a Python-based frontend, allows the direct definition and modification of TensorIR and Relax.

To implement a custom operator, developers define its computation in TE and create IRModules via the BlockBuilder API. BlockBuilder creates primitive functions from TE expressions that are connected via Relax and can be optimized using TensorIR scheduling.⁷ This workflow facilitates efficient execution of specialized operators across diverse hardware platforms while minimizing implementation complexity.

Operating on Tuples. Operators with multiple input tensors are common in neural networks, whereas those producing multiple output tensors are relatively rare. Typically, operators perform a single core computation uniformly across all data. However, PFP introduces a unique requirement, as it maintains separate compute paths and outputs for mean and variance. TVM follows the *one operator = one compute rule* principle, ensuring that each operator executes a single, sequential stream of instructions without divergence. Consequently, logical PFP operations may be split into separate

⁵https://tvm.apache.org/docs/deep_dive/tensor_ir/tutorials/tir_creation.html

⁶https://tvm.apache.org/docs/deep_dive/relax/learning.html

⁷https://mlc.ai/docs/get_started/tutorials/quick_start.html

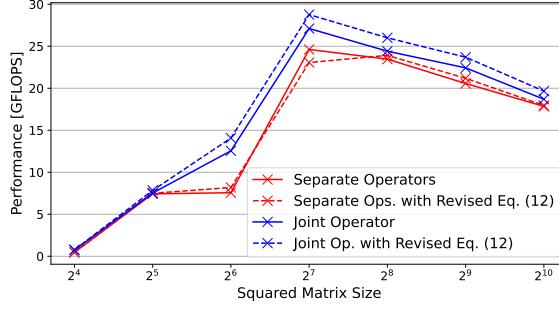


Fig. 5. Performance comparison of operator implementations, evaluating the reformulation from Equation 5 to 12 and the use of separate vs. joint operators for mean and variance paths on a ARM Cortex-A72.

TVM operators, e.g., one for means and another for variances. However, this approach increases the complexity of interconnecting sub-operators, adds overhead, and complicates the implementation of new network architectures. Moreover, from a resource efficiency perspective, this separation is undesirable, as mean and variance calculations share common sub-terms. Achieving this integration requires extensions to TVM’s basic operations such as summation over tuples, however, it enables data reuse and eliminates redundant computations. Figure 5 demonstrates that network architectures employing joint operators, as opposed to separate computational paths for mean and variance, consistently benefit from enhanced data reuse. This structural advantage translates into superior performance across all evaluated cases, indicating the efficacy of the joint formulation in leveraging shared intermediate representations.

Variance and Second Raw Moment. The original formulation of PFP operators is based on mean and variance, with inputs and outputs defined accordingly. However, reformulating Equation 5 to operate on second raw moments for activations and weights enhances data reuse and reduces computational overhead. Using second raw moments, the variance in the PFP dense layer can be computed as:

$$\sigma_{a_i^l}^2 = \sum_{j=1}^{d_{l-1}} \mathbb{E}\left[\left(w_{ij}^l\right)^2\right] \cdot \mathbb{E}\left[\left(x_j^{l-1}\right)^2\right] - \left(\mu_{w_{ij}^l} \cdot \mu_{x_j^{l-1}}\right)^2, \quad (12)$$

thereby reusing the means for the current layer $\mu_{a_i^l}$, the pre-computed second raw moments of the weights $\mathbb{E}[(w_{ij}^l)^2]$ and avoiding conversions from previous activation function outputs $\mathbb{E}[(x_j^{l-1})^2]$, which compute second raw moments by design. The locality of this tuple-based second raw moment operator improves cache efficiency and overall performance. Figure 5 illustrates the performance gains achieved by reformulating Equation 5 to 12 and employing a joint operator.

When consecutive layers differ in their representation of second raw moments and variances, conversion is straightforward using $\mathbb{E}(x^2) = \mu^2 + \sigma^2$. However, converting second raw moments to variances and then back in subsequent layers introduces unnecessary computational overhead. To mitigate this, the operator implementation includes a conversion function as a configurable argument. Ensuring consistency between the inputs and outputs of connected layers remains the responsibility of the model designer. Moreover, the weights need to be expressed accordingly, either as means and variances σ_w^2 (see (7)) or as means and second raw moments $\mathbb{E}[w^2]$ (see (5)).

To minimize redundant computations and facilitate the reuse of intermediate statistical quantities, compute layers—such as dense and convolutional layers—by default expect second raw moments as inputs and produce variances as outputs. Conversely, activation functions expect variances as inputs and produce second raw moments. For architectures

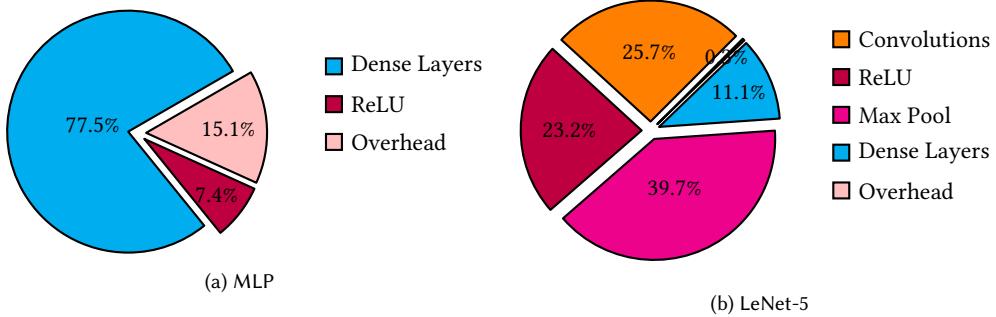


Fig. 6. Execution time distribution across operator types for two PFP-BNN architectures, measured on a Cortex-A72 with a mini-batch size of 10. While dense layers dominate latency in the MLP, LeNet-5 shows a more balanced distribution. Notably, otherwise trivial operators such as ReLU and Max Pool exhibit increased computational cost. The tooling and profiling overhead is typically negligible but becomes noticeable in the low-latency MLP.

composed solely of compute layers and activation functions, this default behavior remains consistent. However, when additional layers are introduced, such as Max Pooling layers that both accept and produce variances (e.g., in LeNet-5), the activation function’s output and the compute layer’s input must be converted to variances.

A special case arises in the *first layer of the network*, where activation uncertainty—and thus variance—is unavailable. In this scenario, Equations 4 and 12 simplify to:

$$\mu_{a_i^l} = \sum_{j=1}^{d_{l-1}} \mu_{w_{ij}^l} \cdot \mu_{x_j^{l-1}} \text{ and } \sigma_{a_i^l}^2 = \sum_{j=1}^{d_{l-1}} \sigma_{w_{ij}^l}^2 \cdot \mu_{x_j^{l-1}}^2. \quad (13)$$

These equations depend on weight variances, requiring the first layer’s weight variances to be stored in variance format. For all subsequent compute layers, weight variance information can be stored directly as second raw moments to reduce computational overhead. Additionally, compute layers support three bias configurations: no bias, deterministic bias, and probabilistic bias with variances.

This section analyzed the integration of custom operators into TVM, emphasizing design considerations specific to PFP operators. The evaluation of different implementation strategies demonstrated that the joint operator approach, combined with the second raw moment formulation, provides the highest efficiency in practice.

6 OPTIMIZING FOR PERFORMANCE

The PFP implementation presented in Section 5 provides a functional BNN framework capable of uncertainty prediction, as described in Section 4, and is already significantly faster than sampling-based BNNs. However, implementation- and hardware-specific safe optimizations can further enhance performance. First, profiling techniques identify the most computationally expensive operators, followed by the application of common optimization strategies to develop a hand-tuned scheduling approach for the MLP. Additionally, we leverage automatic tuning frameworks developed by the TVM community [13, 51, 55, 59]. Finally, we compare PFP implementations across various ARM processor architectures and against SVI-based BNNs.

Table 2. Comparison of manual optimization techniques for the PFP dense operator. Measurements on a Cortex-A72, with a 3-layer MLP and a mini-batch size of 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.

6.1 Profiling Operators

Efficient optimization requires focusing on the most computationally expensive operators, necessitating a detailed analysis of execution costs per operator. TVM provides three execution modes for compiled binaries: normal execution, benchmarking with advanced averaging for precise measurements, and profiling, which reports per-operator latencies. These detailed latency reports enable the evaluation of optimization effects on specific operators, as shown in Table 4, and help visualize time distribution per operator, as illustrated in Figure 6.

Figure 6 illustrates the proportion of execution time spent per operator type. For the MLP, dense layers dominate computational costs, whereas in LeNet-5, latency is more evenly distributed across different layer types. This indicates that operators considered simple in a deterministic setting, such as activation functions or pooling layers, can become computationally complex when operating on distributions. This observation is further supported by detailed per-layer latency benchmarks and profiling results obtained with TVM on an ARM Cortex-A72, as shown in Table 4.

6.2 Manual Optimizations

The MLP is primarily constrained by the latency of dense layers, making the PFP dense operator the most promising target for optimization. We apply the following techniques, commonly used to accelerate matrix-matrix multiplication, to enhance the efficiency of probabilistic dense operators:

- *Tiling*: Partitioning matrices into smaller tiles to optimize memory access patterns. Tile sizes must be tuned.
- *Loop Reordering*: Adjusting loop order to improve vectorization, parallelization, and memory access efficiency.
- *Loop Unrolling*: Unrolling smaller loops to enhance vectorization and reduce loop overhead.
- *Vectorization*: Utilizing SIMD instructions efficiently, requiring careful selection of the appropriate dimension.
- *Parallelization*: Distributing computations across multiple cores while minimizing synchronization and communication overhead.

Table 2 compares the impact of different optimization techniques, evaluating cases where only one optimization is active and where all optimizations except tiling are applied. This allows for an isolated analysis of each technique’s

Table 3. Comparison of Max Pool implementations. Measurements on a Cortex-A72 with a mini-batch size of 10.

Architecture	Implementation	Auto-tuning	Latency	
			Max Pools	Entire Network
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 $k = 2$	No	3.54 ms	21.10 ms
LeNet-5	Vect. Max Pool $k = 2$	All operators	27.28 ms	33.42 ms
LeNet-5	Vect. Max Pool $k = 2$	All except Max Pool	3.69 ms	9.79 ms

effectiveness. For instance, while vectorization is generally a promising optimization, it significantly degrades performance when applied alone, as it relies on a vectorizable inner loop, which must first be established through loop reordering.

Tiling requires a separate evaluation. When applied independently with hand-tuned tile sizes, it proves highly effective. However, it is the only optimization that does not support stochastic tuning. Since the other optimizations benefit from stochastic tuning, enabling tiling disables this option. As a result, applying all optimizations, including tiling but without stochastic tuning, performs worse than basic tiling alone. The best performance is achieved by combining all other optimizations with stochastic tuning while excluding tiling. Thus, the effectiveness of optimizations is assessed without explicit tiling.

Loop unrolling and parallelization emerge as the most critical optimizations for the PFP dense operator. Combining these techniques results in a substantial speedup of 5× compared to the untuned network.

Max Pool Operator. For LeNet-5, the Max Pooling operator by Roth [46] is formulated as a generic reduction, which is suboptimal in performance. We implement a specialized, vectorized Max Pool operator with fixed kernel size for improved efficiency. As evidenced in Table 3, the application of automatically generated schedules fails to enhance performance and, in fact, results in a substantial deterioration in runtime efficiency. Consequently, this hand-optimized operator is excluded from further automatic tuning.

6.3 Automatic Optimizations

While implementing custom schedules for operators remains common, significant advancements in automatic tuning frameworks have been made by the TVM community [13, 51, 55, 59]. The *Meta Scheduler* [51] automates schedule generation, eliminating the need for manual implementations. Conceptually, the Meta Scheduler is a domain-specific probabilistic programming abstraction that explores a large optimization search space. Auto-tuning then navigates this space, benchmarking different implementation variants on the target hardware. While this process is slower than expert-crafted schedules, it typically achieves comparable performance and does not require manual effort from domain experts. Applying all optimizations from Table 2, we achieve nearly identical latencies: 0.743 ms with handwritten schedules and 0.742 ms with the Meta Scheduler. Thus, the Meta Scheduler proves highly effective, even for specialized PFP operators, and is used for further experiments.

Table 4 presents profiling results for the PFP MLP and LeNet-5 before and after tuning, highlighting operator latencies. All operators, except Max Pool layers, benefit significantly from tuning, with dense and convolution layers showing the most substantial acceleration.

Table 4. Profiling of PFP neural architectures to determine most costly layers on a Cortex-A72 with a mini-batch size of 10.

Architecture	Layer	Baseline		Tuned Implementation		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 Network	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 Network	23.698 ms	100.0 %	12.166 ms	100.0 %	1.9×

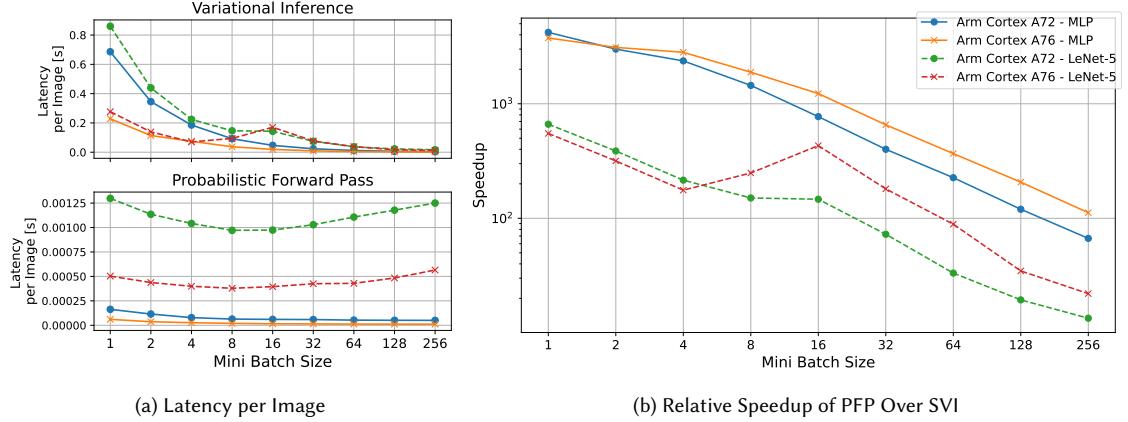
¹ Layers present multiple times in network ² Layers excluded from tuning

Fig. 7. Speedup and latency in relation to mini-batch size. The SVI-BNN, evaluated using 30 samples, exhibits high latency and poor scalability for small mini-batch sizes, resulting in significantly higher per-image latency. In contrast, the PFP implementation maintains consistent performance across all mini-batch sizes due to targeted tuning. The observed speedup highlights the advantages of PFP, particularly for small mini-batch sizes, which are critical for low-latency processing in embedded systems.

6.4 Evaluation and Performance Gain

In resource-constrained embedded systems, real-time and low-latency requirements often necessitate the use of small mini-batch sizes. Figure 7 shows that SVI-based BNNs scale poorly with decreasing mini-batch sizes, incurring substantial computational overhead and high latency per image. To reflect realistic constraints where minimal sample counts are preferred, the SVI-BNN—implemented using Pyro—is evaluated with only 30 samples, which is at the lower bound of

the range (see Figure 1b). In contrast, the PFP implementation is optimized per mini-batch size, maintaining relatively stable latency. Only minor latency increases occur when mini-batch sizes misalign with cache or SIMD instruction sizes.

Overall, the performance gap is substantial, reaching multiple orders of magnitude, as illustrated by the speedups in Figure 7b. For a mini-batch size of 256, speedups range from $13\times$ to $112\times$, while for mini-batch size 1—common in embedded systems—they increase dramatically, reaching $550\times$ to $4200\times$.

Table 5. Algorithm Performance Comparison on Embedded ARM Processors, using vectorized Max Pool and a mini-batch size of 10.

Architecture	batch-size	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	990.2× 901.8×
		Cortex-A72	4.59 ms	0.186 ms	734.74 ms	3.75 ms	0.742 ms	
		Cortex-A76	1.64 ms	0.071 ms	307.52 ms	1.89 ms	0.341 ms	
MLP	100	Cortex-A53	137.80 ms	6.565 ms	-	147.61 ms	15.358 ms	149.6× 255.7×
		Cortex-A72	45.81 ms	1.134 ms	775.32 ms	36.33 ms	5.182 ms	
		Cortex-A76	16.30 ms	0.230 ms	306.89 ms	18.60 ms	1.200 ms	
LeNet-5	10	Cortex-A53	21.14 ms	4.726 ms	-	76.09 ms	35.159 ms	119.4× 205.6×
		Cortex-A72	6.89 ms	0.754 ms	1196.42 ms	21.23 ms	10.022 ms	
		Cortex-A76	3.16 ms	0.347 ms	801.40 ms	9.63 ms	3.897 ms	
LeNet-5	100	Cortex-A53	209.28 ms	41.697 ms	-	801.33 ms	383.680 ms	23.3× 55.3×
		Cortex-A72	70.08 ms	9.524 ms	2708.16 ms	240.94 ms	116.330 ms	
		Cortex-A76	31.51 ms	3.131 ms	2488.73 ms	119.76 ms	45.039 ms	

Table 5 compares probabilistic SVI-based BNNs and PFP-based architectures across multiple embedded ARM processors. Auto-tuning yields clear benefits for both deterministic and PFP implementations.

On average, PFP is $4.4\times$ slower for the MLP and $11.3\times$ slower for LeNet-5 compared to their deterministic counterparts. This slowdown is expected due to the increased computational complexity and the doubling of both parameters and activations. However, compared to a state-of-the-art SVI-based BNN with only 30 samples, PFP achieves substantial average speedups of $574\times$ for the MLP and $101\times$ for LeNet-5.

These results highlight the efficiency gains of an approximate SVI approach combined with code generation, demonstrating the feasibility of deploying such models on resource-constrained embedded systems.

7 CONCLUSION

Bayesian neural networks empower neural networks with probabilistic reasoning, enabling them to issue warnings when confidence in a prediction is low. Distinguishing between aleatoric and epistemic uncertainty allows identification of inherent stochasticity versus uncertainty due to insufficient training data. In safety-critical applications, such as transportation systems, embedding BNNs in devices enables them to trigger warnings under high uncertainty, allowing human operators to intervene when needed. However, despite these advantages, BNNs are rarely deployed in resource-constrained embedded systems due to their substantial computational and memory overhead compared to non-probabilistic models.

This work advances the deployment of probabilistic machine learning models on embedded systems. We demonstrate how a Stochastic Variational Inference-based BNN can be efficiently executed on embedded ARM CPUs by leveraging the Probabilistic Forward Pass approximation. To achieve this, we extend TVM, a deep learning compiler, to support essential probabilistic operators and optimize them for target hardware architectures.

The Probabilistic Forward Pass mitigates the primary computational bottleneck of BNNs—the need for multiple forward passes per prediction—by assuming Gaussian-distributed weights and activations, enabling a single analytical forward pass. This approach extends the core assumption of SVI-based BNNs, which model weights as parametric distributions, to the activation domain. While this limits the network’s ability to capture complex, non-Gaussian activation distributions, it significantly reduces computational costs when Gaussian approximations are sufficient.

A key advantage of PFP is its compatibility with SVI-trained BNNs, enabling seamless conversion. Our comparative analysis on MNIST, Ambiguous-MNIST, and Fashion-MNIST shows that both approaches achieve comparable predictive performance. While the SVI baseline slightly outperforms PFP in distinguishing aleatoric from epistemic uncertainty, the latter remains equally effective in out-of-domain image detection.

Efficient hardware implementation is essential for deploying probabilistic models on embedded systems. Since they rely on specialized operators for Gaussian distributions—unsupported by standard vendor libraries for machine learning acceleration—we utilize the deep learning compiler TVM. Its flexibility in implementing and optimizing custom operators across hardware architectures makes it an ideal choice.

Our evaluation of multiple implementation variants shows that joint operators, which compute mean and variance in a single operation, yield greater efficiency. Further optimizations, such as eliminating unnecessary variance conversions and simplifying the first network operators for deterministic inputs, further reduce computational costs.

Following manual optimizations, additional performance gains were achieved through profiling-based operator tuning. We applied targeted optimizations to probabilistic dense operators, developed a more efficient Max Pool operator, and leveraged TVM’s Meta Scheduler for auto-tuning. As a result, our optimized Probabilistic Forward Pass implementation achieved a two-order-of-magnitude speedup over the SVI-based BNN baseline on ARM processors. For small mini-batch sizes, which are critical for low-latency embedded applications, speedups of up to 4200 \times were achieved.

To our knowledge, this work presents the first end-to-end demonstration of training, optimization, and deployment of Bayesian neural networks on resource-constrained embedded systems. Our findings illustrate that integrating Bayesian approximations with deep learning compilers enables the deployment of otherwise computationally prohibitive probabilistic models. We hope this work marks the beginning of further research bridging the gap between resource-intensive probabilistic models and constrained embedded hardware—bringing uncertainty-aware neural networks into everyday devices and enabling them to acknowledge uncertainty by saying, “I don’t know.”

ACKNOWLEDGMENTS

The authors gratefully acknowledge the financial support 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 supported 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), and the federal states of Styria, Upper Austria and Tyrol.

REFERENCES

- [1] Farhan Abrol, Stephan Mandt, Rajesh Ranganath, and David Blei. 2014. Deterministic annealing for stochastic variational inference. *stat* 1050 (2014), 7. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=5cee21ae4605a330d9977164523be4b865df6ebd>
- [2] Hiromitsu Awano and Masanori Hashimoto. 2023. B2N2: Resource efficient Bayesian neural network accelerator using Bernoulli sampler on FPGA. *Integration* 89 (2023), 1–8. <https://doi.org/10.1016/j.vlsi.2022.11.005>

- [3] Subho S. Banerjee, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. 2019. AcMC2: Accelerating Markov Chain Monte Carlo Algorithms for Probabilistic Models. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 515–528. <https://doi.org/10.1145/3297858.3304019>
- [4] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. 2019. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research* 20, 28 (2019), 1–6. <http://jmlr.org/papers/v20/18-403.html>
- [5] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. 2017. Variational inference: A review for statisticians. *J. Amer. Statist. Assoc.* 112, 518 (2017), 859–877. <https://doi.org/10.1080/01621459.2017.1285773>
- [6] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. 2015. Weight Uncertainty in Neural Networks. In *International Conference on Machine Learning*, Vol. 37. PMLR. <https://proceedings.mlr.press/v37/blundell15.html>
- [7] Djohan Bonnet, Tifenn Hirtzlin, Atreya Majumdar, Thomas Dalgaty, Eduardo Esmamhotto, Valentina Meli, Niccolo Castellani, Simon Martin, Jean-François Nodin, Guillaume Bourgeois, Jean-Michel Portal, Damien Querlioz, and Elisa Vianello. 2023. Bringing uncertainty quantification to the extreme-edge with memristor-based Bayesian neural networks. *Nature Communications* 14, 1 (Nov 2023), 7530. <https://doi.org/10.1038/s41467-023-43317-9>
- [8] Steve Brooks, Andrew Gelman, Galin L. Jones, and Xiao-Li Meng. 2011. *Handbook of Markov Chain Monte Carlo*. CRC Press. <https://doi.org/10.1201/b10905>
- [9] 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. 2024. Probabilistic photonic computing with chaotic light. *Nature Communications* 15, 1 (dec 2024), 10445. <https://doi.org/10.1038/s41467-024-54931-6>
- [10] Han Cai, Ligeng Zhu, and Song Han. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. (feb 2019). arXiv:1812.00332 [cs.LG]
- [11] Ruizhe Cai, Ao Ren, Ning Liu, Caiwen Ding, Luhao Wang, Xuehai Qian, Massoud Pedram, and Yanzhi Wang. 2018. VIBNN: Hardware Acceleration of Bayesian Neural Networks. *SIGPLAN Not.* 53, 2 (mar 2018), 476–488. <https://doi.org/10.1145/3296957.3173212>
- [12] Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. ZeroQ: A Novel Zero Shot Quantization Framework. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Seattle, WA, USA, 13166–13175. <https://doi.org/10.1109/CVPR42600.2020.01318>
- [13] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: an automated end-to-end optimizing compiler for deep learning. In *USENIX Conference on Operating Systems Design and Implementation (OSDI)*. USENIX Association, USA, 579–594. <https://dada.cs.washington.edu/research/tr/2017/12/UW-CSE-17-12-01.pdf>
- [14] Stefan Depeweg, Jose-Miguel Hernandez-Lobato, Finale Doshi-Velez, and Steffen Udluft. 2018. Decomposition of Uncertainty in Bayesian Deep Learning for Efficient and Risk-sensitive Learning. In *International Conference on Machine Learning*. PMLR, 1184–1193. <https://proceedings.mlr.press/v80/depeweg18a.html>
- [15] Francesco D' Angelo and Vincent Fortuin. 2021. Repulsive Deep Ensembles are Bayesian. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 3451–3465. https://proceedings.neurips.cc/paper_files/paper/2021/file/1c63926ebcabda26b5cdb31b5cc91efb-Paper.pdf
- [16] Sebastian Farquhar, Lewis Smith, and Yarin Gal. 2020. 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*. <https://bayesiandeeplearning.org/2019/papers/45.pdf>
- [17] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern recognition letters* 27, 8 (2006), 861–874. <https://doi.org/10.1016/j.patrec.2005.10.010>
- [18] Siyuan Feng, Bohan Hou, Hongyi Jin, Wuwei Lin, Junru Shao, Ruihang Lai, Zihao Ye, Lianmin Zheng, Cody Hao Yu, Yong Yu, and Tianqi Chen. 2023. TensorIR: An Abstraction for Automatic Tensorized Program Optimization. In *International Conference on Architectural Support for Programming Languages and Operating Systems* (Vancouver, BC, Canada) (ASPLOS). Association for Computing Machinery, New York, NY, USA, 804–817. <https://doi.org/10.1145/3575693.3576933>
- [19] Yarin Gal and Zoubin Ghahramani. 2016. Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference. arXiv:1506.02158 [stat.ML]
- [20] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a Bayesian approximation: representing model uncertainty in deep learning. In *International Conference on International Conference on Machine Learning (ICML)*. PMLR, 1050–1059. <https://proceedings.mlr.press/v48/gal16.html>
- [21] Laura I Galindez Olascoaga, Wannes Meert, Nimish Shah, Marian Verhelst, and Guy Van den Broeck. 2019. Towards Hardware-Aware Tractable Learning of Probabilistic Models. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/e77910ebb93b511588557806310f78f1-Paper.pdf
- [22] Zoubin Ghahramani. 2015. Probabilistic machine learning and artificial intelligence. *Nature* 521, 7553 (2015), 452–459. <https://doi.org/10.1038/nature14541>
- [23] Song Han, Huiyi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *International Conference on Learning Representations ICLR*, Yoshua Bengio and Yann LeCun (Eds.). arXiv:1510.00149 [cs.CV]

- [24] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2019. 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
- [25] Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. 2013. Stochastic variational inference. *Journal of Machine Learning Research* 14, 1 (May 2013), 1303–1347. <http://jmlr.org/papers/v14/hoffman13a.html>
- [26] Matthew D Hoffman, Andrew Gelman, et al. 2014. The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research* 15, 1 (2014), 1593–1623. <https://www.jmlr.org/papers/volume15/hoffman14a/hoffman14a.pdf>
- [27] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. https://openaccess.thecvf.com/content_cvpr_2018/html/Jacob_Quantization_and_Training_CVPR_2018_paper.html
- [28] Laurent Valentin Jospin, Hamid Laga, Farid Boussaid, Wray Buntine, and Mohammed Bennamoun. 2022. Hands-On Bayesian Neural Networks—A Tutorial for Deep Learning Users. *IEEE Computational Intelligence Magazine* 17, 2 (2022), 29–48. <https://doi.org/10.1109/MCI.2022.3155327>
- [29] Alex Kendall and Yarin Gal. 2017. What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? *Advances in Neural Information Processing Systems* 30 (2017), 5574–5584. https://proceedings.neurips.cc/paper_files/paper/2017/file/2650d6089a6d640c5e85b2b88265dc2b-Paper.pdf
- [30] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. (2014). arXiv:1412.6980 [cs.LG]
- [31] Diederik P Kingma and Max Welling. 2013. Auto-Encoding Variational Bayes. *CoRR* (2013). <https://doi.org/10.48550/arXiv.1312.6114>
- [32] Torben Krieger, Bernhard Klein, and Holger Fröning. 2023. Towards Hardware-Specific Automatic Compression of Neural Networks. *International Workshop on Practical Deep Learning in the Wild at the AAAI Conference on Artificial Intelligence* (2023). arXiv:2212.07818 [cs.LG]
- [33] Ruihang Lai, Junru Shao, Siyuan Feng, Steven S. Lyubomirsky, Bohan Hou, Wuwei Lin, Zihao Ye, Hongyi Jin, Yuchen Jin, Jiawei Liu, Lesheng Jin, Yaxing Cai, Ziheng Jiang, Yong Wu, Sunghyun Park, Prakalp Srivastava, Jared G. Roesch, Todd C. Mowry, and Tianqi Chen. 2023. Relax: Composable Abstractions for End-to-End Dynamic Machine Learning. (2023). arXiv:2311.02103
- [34] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in Neural Information Processing Systems* 30 (2017), 6405–6416. https://proceedings.neurips.cc/paper_files/paper/2017/file/9ef2ed4b7fd2c810847ffa5fa85bce38-Paper.pdf
- [35] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In *IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2–14. <https://doi.org/10.1109/CGO51591.2021.9370308>
- [36] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. <https://doi.org/10.1109/5.726791>
- [37] Jelin Leslin, Antti Hyttinen, Karthekeyan Periasamy, Lingyun Yao, Martin Trapp, and Martin Andraud. 2022. A Hardware Perspective to Evaluating Probabilistic Circuits. In *International Conference on Probabilistic Graphical Models*, Vol. 186. PMLR, 349–360. <https://proceedings.mlr.press/v186/leslin22a.html>
- [38] Xingchen Li, Bingzhe Wu, Guangyu Sun, Zhe Zhang, Zhihang Yuan, Runsheng Wang, Ru Huang, Dimin Niu, Hongzhong Zheng, Zhichao Lu, Liang Zhao, Meng-Fan Marvin Chang, Tianchan Guan, and Xin Si. 2022. Enabling High-Quality Uncertainty Quantification in a PIM Designed for Bayesian Neural Network. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 1043–1055. <https://doi.org/10.1109/HPCA53966.2022.00008>
- [39] Yudeng Lin, Qingtian Zhang, Jianshi Tang, Bin Gao, Chongxuan Li, Peng Yao, Zhengwu Liu, Jun Zhu, Jiwu Lu, Xiaobo Sharon Hu, He Qian, and Huaqiang Wu. 2019. Bayesian Neural Network Realization by Exploiting Inherent Stochastic Characteristics of Analog RRAM. In *IEEE International Electron Devices Meeting (IEDM)*, 14.6.1–14.6.4. <https://doi.org/10.1109/IEDM19573.2019.8993616>
- [40] Christos Louizos, Karen Ullrich, and Max Welling. 2017. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 30. Curran Associates Inc., 3290–3300. https://proceedings.neurips.cc/paper_files/paper/2017/file/69d1fc78dbda242c43ad6590368912d4-Paper.pdf
- [41] Jishnu Mukhoti, Andreas Kirsch, Joost van Amersfoort, Philip H. S. Torr, and Yarin Gal. 2022. *Deep Deterministic Uncertainty: A Simple Baseline*. Technical Report. arXiv:2102.11582 [cs.LG]
- [42] Kevin P Murphy. 2012. *Machine learning: a probabilistic perspective*. MIT press. <https://dl.acm.org/doi/abs/10.5555/2380985>
- [43] Radford M. Neal. 1996. *Bayesian Learning for Neural Networks*. Vol. 118. Springer Science & Business Media. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=db869fa192a3222ae4f2d766674a378e47013b1b>
- [44] Radford M Neal. 2011. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo* 2, 11 (2011), 2–11. <https://www.dam.brown.edu/people/geman/Homepage/CV/HandbookChapter5.pdf>
- [45] Jared Roesch, Steven Lyubomirsky, Logan Weber, Josh Pollock, Marisa Kirisame, Tianqi Chen, and Zachary Tatlock. 2018. Relay: a new IR for machine learning frameworks. In *SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL)*. Association for Computing Machinery, New York, NY, USA, 58–68. <https://doi.org/10.1145/3211346.3211348>
- [46] Wolfgang Roth. 2021. *Probabilistic Methods for Resource Efficiency in Machine Learning*. Ph. D. Dissertation. Graz University of Technology Austria. <https://download.spsc.tugraz.at/thesis/thesis-wroth.pdf>
- [47] Wolfgang Roth and Franz Perlmutter. 2016. Variational inference in neural networks using an approximate closed-form objective. In *NIPS Workshop on Bayesian Deep Learning*. http://bayesanddeeplearning.org/2016/papers/BDL_13.pdf

- [48] Wolfgang Roth, Günther Schindler, Bernhard Klein, Robert Peharz, Sebastian Tschiatschek, Holger Fröning, Franz Pernkopf, and Zoubin Ghahramani. 2024. Resource-Efficient Neural Networks for Embedded Systems. *Journal of Machine Learning Research* 25, 50 (2024), 1–51. <http://jmlr.org/papers/v25/18-566.html>
- [49] Günther Schindler, Matthias Zöhrer, Franz Pernkopf, and Holger Fröning. 2018. Towards Efficient Forward Propagation on Resource-Constrained Systems. In *Machine Learning and Knowledge Discovery in Databases (Lecture Notes in Computer Science, Vol. 11051)*, Michele Berlingero, Francesco Bonchi, Thomas Gärtner, Neil Hurley, and Georgiana Ifrim (Eds.). Springer, 426–442. https://doi.org/10.1007/978-3-030-10925-7_26
- [50] Claude Elwood Shannon. 1948. A mathematical theory of communication. *The Bell system technical journal* 27, 3 (1948), 379–423. <https://doi.org/10.1002/j.1538-7305.1948.tb01338.x>
- [51] Junru Shao, Xiyu Zhou, Siyuan Feng, Bohan Hou, Ruihang Lai, Hongyi Jin, Wuwei Lin, Masahiro Masuda, Cody Hao Yu, and Tianqi Chen. 2022. Tensor Program Optimization with Probabilistic Programs. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 35783–35796. https://proceedings.neurips.cc/paper_files/paper/2022/file/e894eafae43e68b4c8dfdacf742bcbf3-Paper-Conference.pdf
- [52] Mrinank Sharma, Sebastian Farquhar, Eric Nalisnick, and Tom Rainforth. 2023. Do Bayesian Neural Networks Need To Be Fully Stochastic?. In *International Conference on Artificial Intelligence and Statistics*, Francisco Ruiz, Jennifer Dy, and Jan-Willem van de Meent (Eds.), Vol. 206. PMLR, 7694–7722. <https://proceedings.mlr.press/v206/sharma23a.html>
- [53] Sophie Steger, Christian Knoll, Bernhard Klein, Holger Fröning, and Franz Pernkopf. 2024. Function Space Diversity for Uncertainty Prediction via Repulsive Last-Layer Ensembles. In *ICML Workshop on Structured Probabilistic Inference & Generative Modeling*. <https://openreview.net/forum?id=FbMN9HJgHI>
- [54] Qiyu Wan, Haojun Xia, Xingyao Zhang, Lening Wang, Shuaiwen Leon Song, and Xin Fu. 2021. Shift-BNN: Highly-Efficient Probabilistic Bayesian Neural Network Training via Memory-Friendly Pattern Retrieving. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Association for Computing Machinery, New York, NY, USA, 885–897. <https://doi.org/10.1145/3466752.3480120>
- [55] Xingfu Wu, Praveen Paramasivam, and Valerie Taylor. 2023. 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, 29–35. <https://doi.org/10.1145/3624062.3626079>
- [56] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR* (2017). arXiv:1708.07747 [cs.LG]
- [57] Lingyun Yao, Martin Trapp, Jelin Leslin, Gaurav Singh, Peng Zhang, Karthekeyan Periasamy, and Martin Andraud. 2024. On hardware-efficient inference in probabilistic circuits. In *Conference on Uncertainty in Artificial Intelligence (UAI)*. arXiv:2405.13639 [cs.LG]
- [58] Cheng Zhang, Judith Bütépage, Hedvig Kjellström, and Stephan Mandt. 2019. Advances in Variational Inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41, 8 (2019), 2008–2026. <https://doi.org/10.1109/TPAMI.2018.2889774>
- [59] Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez, and Ion Stoica. 2020. Ansor: Generating High-Performance Tensor Programs for Deep Learning. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 863–879. <https://www.usenix.org/conference/osdi20/presentation/zheng>