

# Memoization: Accelerating MCD BNN by Attribution-Based Dynamic Precision Scaling

Huiwon Choi\*, Myoungsoo Jung\*

*\*Computer Architecture and Memory Systems Laboratory, KAIST*

## Abstract

Monte-Carlo Dropout Bayesian Neural Network (MCD BNN) measures the uncertainty of inference result, thereby ensuring safer deployment of neural networks in real-world applications. However, since MCD BNN requires a large number of stochastic forward passes per inference, it suffers from significantly higher inference overhead compared with conventional neural network.

We propose the Memoization, an algorithm-architecture co-design method composed of 1) an Attribution-based Dynamic Approximation algorithm and 2) a domain-specific hardware accelerator tailored for the algorithm. This co-optimization enables fast, energy-efficient and accurate MCD BNN inference. To the best of our knowledge, Memoization is the first to compute input-specific attribution at inference time for estimating neuron importance.

We evaluated Memoization algorithm in terms of accuracy on MCD BNN and confirm Memoization PE's effectiveness on a systolic array. Our evaluation shows that Memoization reduces the latency to 30% of the baseline, with less than a 1% drop in accuracy.

## 1 Introduction

Machine learning (ML) models have achieved remarkable success across a wide range of domains. However, their predictions can sometimes be unreliable, which can lead to serious consequences in real-world applications. To address this issue, MCD BNNs [1] have been increasingly employed in an safety-critical applications which requires reliable and robust inference, such as healthcare, autonomous driving, and robotics. MCD BNNs enhance the safe use of ML by providing uncertainty estimates, allowing users to judge whether an inference result is confident or potentially unreliable [2], [3]. For example, in healthcare applications, predictions with high uncertainty can be excluded from diagnoses and deferred to human doctor for final assessment. To quantify the uncertainty of inference result, MCD BNN performs multiple stochastic forward passes for each input. However, this requirement significantly increases the inference overhead compared to conventional neural networks.

To address the inference overhead in neural networks, prior studies have proposed Importance-based Dynamic Precision Scaling (IDPS) [4], [5], [6]. IDPS exploits the observation that, during inference, neurons have different levels of importance and the importance varies across inputs. Therefore, dynamically identifying neuron importance allows less important neurons to be computed with reduced precision. Using this, IDPS divides the inference procedure into two phases: (1) The prediction stage, which estimates the importance of each neuron for the given input. (2) The execution stage, which executes computations in full precision for highly important neurons while use reduced precision for less important neurons. Most existing IDPS approaches prioritize minimizing the overhead of the prediction stage rather than achieving more accurate importance estimation. However, this design choice can limit the achievable approximation ratio at the execution stage, thereby reducing the computational gains.

To this end, we propose Memoization, which is an algorithm-architecture co-design approach that accurately estimates neuron importance using attribution values while incurring only a small prediction overhead, thereby maximizing inference efficiency of MCD BNNs. At the algorithm level, we present an Attribution-based IDPS algorithm, referred to as the Memoization-Algorithm. The Memoization-Algorithm exploits the characteristics of MCD BNNs to achieve accurate neuron-importance estimation using attribution values, while keeping the prediction overhead low by reusing the predicted importance across multiple stochastic forward passes.

At the hardware level, we propose Memoization-Hardware, which is designed to efficiently support the distinctive two-stage operations of the Memoization Algorithm. Memoization-Hardware consists of (1) an adder-tree-based PE capable of efficiently handling both high- and low-precision computations, and (2) a systolic-array-based dataflow that maximizes data reuse during mixed-precision execution. During the Prediction stage, the adder-tree-based PE operates as a full-precision multiplier to compute each neuron's attribution value. During the Execution stage, the PE operates as a high-precision computations for the selected important neurons and as a low-precision accumulator for the remaining neurons, using proposed dataflow.

The main contribution can be summarized as follow :

- Analysis of MCD BNN characteristics. We analyze the inference characteristics of MCD BNNs and show that importance distribution across stochastic forward passes is highly consistent.
- Attribution based runtime importance estimation. The proposed Memoization-Algorithm is the first to compute input-specific attribution at inference time for importance estimation, achieving a high approximation ratio while preserving inference accuracy.
- Practical hardware design. This work shows that both high- and low-precision MAC operations can be supported by common Processing Element, enabling Memoization-hardware to remain effective across a wide range of approximation ratios required by Memoization-Algorithm.

Our evaluation shows that Memoization can shorten the inference latency of the MCD-BNN to 30% of that of the baseline systolic-array-based system, with less than a 1% drop in accuracy.

## 2 Background and Motivation

### 2.1 Bayesian Neural Network

**Bayesian Neural Network** Conventional deterministic inference often fails to make correct predictions on out-of-distribution (OOD) data—data not seen during training—thereby posing a significant risk of prediction errors. In contrast, Bayesian inference [7] can identify whether an input is OOD and, by doing so, mathematically quantify the uncertainty of its predictions. This allows users to recognize when its prediction is unreliable, thereby enabling the safer use of neural networks even when perfect prediction is impossible. However, due to the high dimensionality of modern NNs, it is infeasible to compute uncertainty exactly in Bayesian inference, which has motivated the development of various approximate inference methods. Among these, Monte-Carlo Dropout (MCD) BNNs have become one of the most widely used approximation techniques for Bayesian inference.

**Monte-Carlo Dropout BNN** MCD BNNs [1] modify the conventional deterministic forward pass into a stochastic forward pass by inserting dropout layers, which randomly deactivate a fraction of the input tensor, into the backbone neural network and activating them during inference. By conducting multiple stochastic forward passes for a single input, one can obtain the variance of the outputs, which can be used to approximate the prediction uncertainty. Since the output variance is required to properly approximate uncertainty, it is essential to perform a sufficient number of stochastic forward passes for each input. Nevertheless, this inference scheme introduces significantly heavier inference overhead than conventional neural networks, which perform only a single deterministic forward pass per input.

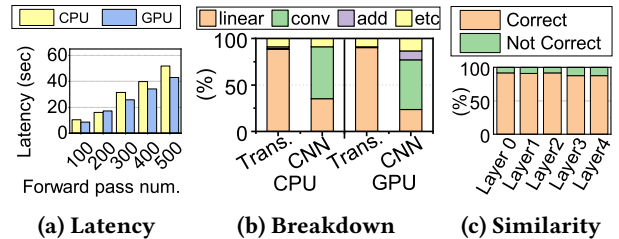


Figure 1: MCD BNN analysis.

**Inference Overhead of MCD BNN** To be precise, we analyze the inference overhead of MCD BNN models with Transformer [8] and CNN [9] backbones on the ImageNet dataset [10]. MCD BNN and conventional inference differs in two key aspects. It requires multiple stochastic forward passes per input and activates dropout during inference. Figure 1a illustrates that the inference time increases linearly with the number of stochastic forward passes per input compared to conventional inference. Figure 1b decomposes the end-to-end inference latency of MCD BNNs and shows that the contribution of dropout layers is negligible, while the MAC-dominated layers (e.g., convolution and linear layers) remain the primary bottleneck as in conventional inference. These results indicate that the difference between inference overhead of MCD BNNs and non-BNNs is driven by the number of stochastic forward passes whereas dropout itself has little impact on the overall computational characteristics.

To address this overhead, domain-specific accelerators for MCD BNNs have been actively studied, improving computational efficiency 1) by applying static optimizations tailored to dropout granularity [11], [12], [13], [14] or 2) by exploiting the dynamic sparsity introduced by ReLU [15]. However, they either 1) fail to leverage dynamical optimization opportunities that depend on input characteristics or 2) suffer from limited scalability across diverse model architectures.

### 2.2 IDPS Algorithm

**Importance-based Dynamic Precision Scaling** IDPS reduces inference overhead while maintaining accuracy, by leveraging the input-dependent fluctuation in neuron importance. This allows more fine-grained optimization than static methods such as uniform quantization which applies the same quantization across the whole model without considering input-specific characteristics at runtime [4], [6], [16]. As illustrated in Fig. 3, IDPS processes each layer in a prediction-execution manner. In the prediction stage, all neurons are processed using low-precision computation. Based on these low-precision results, neuron importance is determined using criteria such as the magnitude of these results. An importance mask is then generated to indicate the importance of each neuron for the subsequent execution stage. The mask is represented as a bitmap, where important neurons

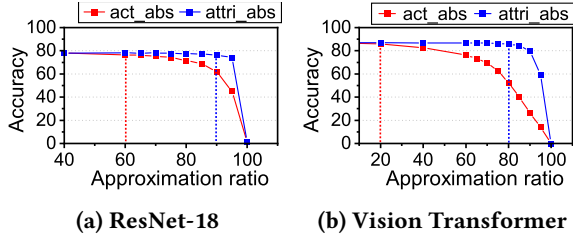


Figure 2: Importance-based Dynamic Precision scaling.

are marked as 1 and unimportant neurons as 0. In the execution stage, full-precision computation is performed only on the neurons marked as important. Then the results from high-precision computation of the important neurons are combined with the low-precision results of the unimportant neurons, and passed as input to the next layer, where the same procedure is repeated.

**Significance of Neuron Importance Criteria** In IDPS, accurately identifying important neurons is crucial for maximizing the ratio of neurons that can be approximated while preserving inference accuracy. To demonstrate this, we analyze the trade-off between inference accuracy drop and approximation ratio using two widely adopted vision models, ResNet-18 [17] and Vision Transformer [8], on the ImageNet dataset [10]. For each input, we pre-profiled neuron importance according to a given criterion to generate an importance mask and then performed inference by computing important neurons in full precision and the remaining neurons in low precision.

Figure 2 presents the results when the importance mask is derived using either the activation magnitude (act\_abs) or attribution values (attri\_abs). The attribution values leverages Integrated Gradients (IG) [18], a technique used to precisely quantify the contribution of neurons to the model’s predictions. As shown in the Figure 2, in ResNet-18, act\_abs and attri\_abs achieve approximation ratios of up to 60% and 90%, respectively, while keeping the accuracy drop below 1%. In the Vision Transformer, the corresponding ratios are 20% and 80%, respectively. These results highlight that determining neuron importance using attribution enables a significantly higher degree of approximation without sacrificing accuracy. However, attribution cannot be directly adopted as the importance criterion in practical IDPS, as computing attribution requires backpropagation, which incurs substantial runtime overhead.

### 3 Memoization Algorithm

As discussed earlier, while using attribution as the importance criterion in IDPS allows a larger fraction of neurons to be processed in low precision, it is not practical in conventional deterministic inference due to the high overhead

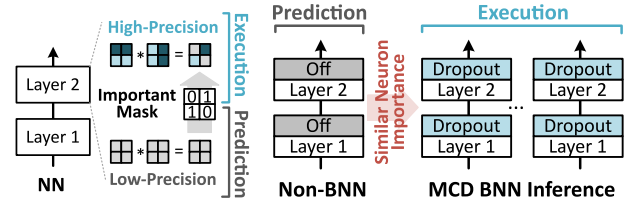


Figure 3: IDPS Algo. Figure 4: Memoization Algo.

of computing attribution. In conventional deterministic inference, computation is completed with a single forward pass. However, determining input-specific attribution requires both forward propagation and backward propagation, whose cost outweighs the potential computational savings from approximation. In contrast, MCD BNNs perform multiple forward passes for the same input with identical weights. We observe that neuron importance remains highly consistent across these stochastic passes, which enables a novel IDPS approach leveraging attribution.

**Similarity in Stochastic Forward Passes** Since MCD BNN performs multiple stochastic forward passes for the same input using identical weights in a single inference, we observe that the neuron importance identified across these passes remains highly similar. This behavior arises because, for in-distribution data, enabling dropout does not alter the feature-extraction process determined by the input and weights [15]. Rather, it only introduces minor perturbations.

Figure 5 visualizes the neuron importance for each layer in AlexNet [19] when evaluated on the same input under two settings: with dropout disabled (deterministic inference) and enabled (stochastic inference). Neurons in the top 20% by importance are shown in white, while the rest are shown in black. The figure shows that both modes exhibit similar neuron importance from layer 1 to deeper layers.

To quantify this observation, Figure 1c compares neuron importance between dropout disabled and enabled, using AlexNet [19] as the backbone on the ImageNet dataset [10]. A neuron is considered Correct if both modes consistently classify it as either important or unimportant. As shown in Figure 1c, nearly 90% of neurons are classified as Correct, confirming the high similarity in neuron importance between the two inference modes.

**Attribution-based IDPS for MCD BNN** Building on this observation, we propose the Memoization Algorithm, which

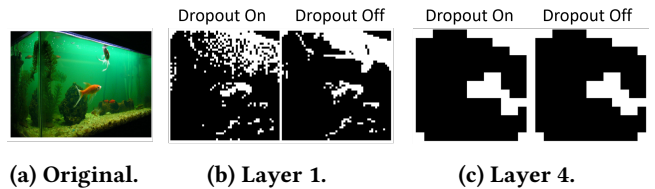


Figure 5: Similarity of Forward Passes in MCD BNN.

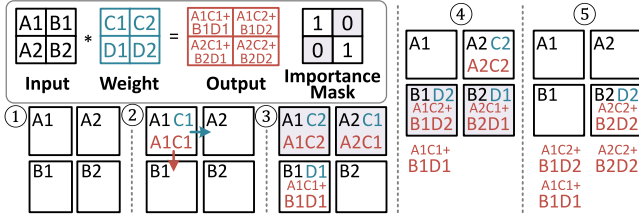


Figure 6: Input Stationary Systolic Array.

reuses neuron importance predicted from a single non-BNN forward pass for all stochastic forward passes in MCD BNN inference. As illustrated in Figure 4, the algorithm operates as follows: First, we profile the model offline to determine the maximum tolerable approximation ratio, denoted by  $\alpha$ . Second, at runtime, we perform inference using the Prediction-Execution stages. During the Prediction stage, we perform one deterministic forward and backward pass to compute neuron attributions for the given input and generate an importance mask, where neurons whose attribution values rank in the top  $\alpha\%$  are marked as important. During the Execution stage, this importance mask is applied across all stochastic forward passes, enabling approximate computation for unimportant neurons and full-precision computation for important ones.

Since the predicted neuron importance is reused across all stochastic forward passes, the amortized prediction overhead per pass becomes negligible, while the use of attribution-based importance yields larger computational gains in the Execution stage. Furthermore, because attribution can be computed via backpropagation for any differentiable model, Memoization Algorithm is model-agnostic and does not rely on structural properties such as ReLU or MaxP, thus making it applicable to a wide range of neural network architectures. **Approximation method** Our algorithm performs low precision multiplication by eliminating the mantissa of weight operand. Generally, neural networks represents weights and inputs in floating-point format. Floating point format consists of a sign  $S$ , an exponent  $E$  and a mantissa  $M$ , where the actual multiplication is performed on the mantissa.

$$I \times W = (S_I \oplus S_W) \cdot (1.M_I \times 1.M_W) \cdot 2^{(E_I+E_W)} \quad (1)$$

By eliminating the mantissa of weight operand, floating point multiplication can be replaced with addition and shift operations, thereby reducing computational overhead. Moreover, since the mantissa of weights for unimportant neurons does not need to be fetched, memory bottlenecks can be alleviated [20]. Beyond these efficiency benefits, eliminating the mantissa is also particularly well suited for the Memoization algorithm, as it does not require additional steps to combine low- and high precision results across unimportant and important neurons and preserves the output variance of MCD BNN by not restricting the range of representable values.

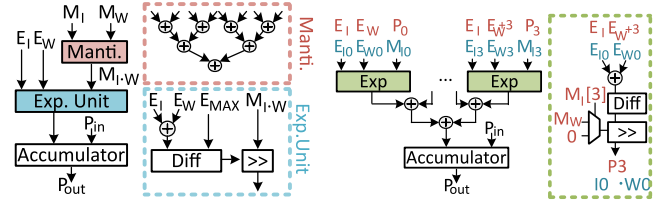


Figure 7: Floating-point PE. Figure 8: Adder-tree PE.

## 4 Memoization Hardware

If hardware can effectively support the performance enhancement of the Memoization Algorithm, the performance of MCD BNN can be greatly improved. As shown in Figure 2a, non-important neurons account for about 90% of the operations in ResNet-18 inference, enabling roughly an 86% reduction in mantissa multiplications in end-to-end MCD-BNN inference with 50 stochastic forward passes per inference [21], even after accounting for the additional cost caused by the Prediction stage.

**A Naive Hardware Baseline** As a baseline hardware, we consider a systolic array composed of floating-point PEs, similar to those used in existing Neural Network accelerators such as TPUs [22].

Figure 6 illustrates the conceptual design of the systolic array. To compute the output matrix by multiplying an input matrix with a weight matrix, a systolic array [23] with input-stationary dataflow operates as follows: the input values are first stored in each PE (①), and the weights are then injected into the array in a diagonal pattern. Upon receiving a weight, each PE multiplies it with the locally stored input value and adds the result to the partial sum received from the PE above. As the computation proceeds, each PE forwards the weight to the PE on its right and propagates the partial sum downward. (② ~④) Through this process, the final outputs emerge at the bottom of the array in a diagonal pattern. (⑤)

Inside each PE, a multiply-accumulate (MAC) operation is performed, where the input is denoted as  $I$ , the weight as  $W$ , and the accumulated partial sum as  $PSUM$ . The operation can be expressed as (2).

$$PSUM_{out} = I * W + PSUM_{in} \quad (2)$$

As shown in Figure 7, a floating-point PE [24], [25] consists of a mantissa module, an exponent module and an accumulator. The mantissa module multiplies the mantissas of the input and the weight. Since mantissa multiplication requires the accumulation of multiple partial products, this module is composed of several adders, such as a Wallace tree or other adder-tree structures. The exponent module first adds the exponents of the input and the weight, determines difference between this sum and that of the partial sum, and then aligns the mantissa multiplication result accordingly. This functionality is implemented with components such as

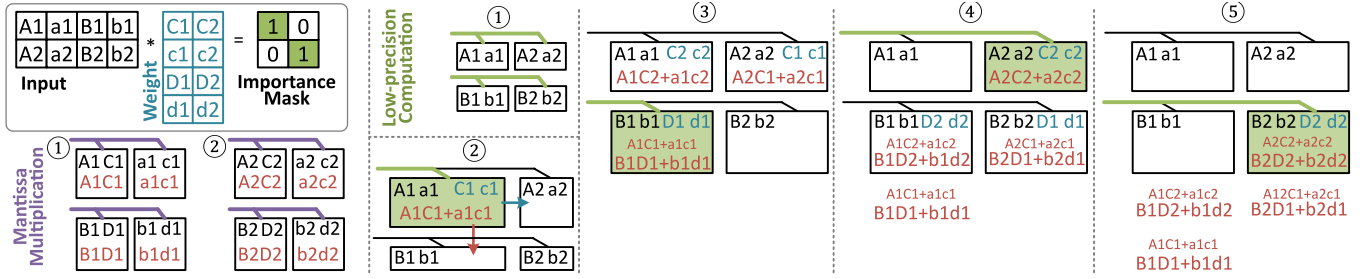


Figure 9: The Proposed Dataflow.

adders, subtractors, and shifters. Finally, at accumulator, the aligned mantissa is added to the partial sum, and a rounding step is applied to produce the final MAC result.

The Memoization Algorithm can be executed on this baseline hardware as follows. During the Prediction stage, the accelerator simply performs forward and backward propagation in high precision, which can be decomposed into floating point matrix multiplications and thus executed in the same manner as with a conventional systolic array. In contrast, the Execution stage requires each neuron to be computed with different precision levels, dynamically determined by the importance mask. For neurons identified as important, high-precision computation is performed, utilizing both the mantissa and exponent modules. But as shown by the purple in Figure 6, mantissa multiplication is skipped for less important neurons, and only the exponent module is used.

#### 4.1 Adder-tree based PE

**Challenge** While the Prediction stage can be efficiently processed on the baseline hardware, the Execution stage cannot be handled as effectively because the mantissa module remains largely under-utilized. As a result, the reduction in arithmetic operations does not translate into a corresponding reduction in latency, providing only limited benefits such as power gating. To overcome this limitation, we propose a new PE design that can effectively utilize the mantissa module even during low-precision computations.

**Adder-tree based PE** The adder-tree-based PE leverages the observation that the adder tree, which composes the mantissa module, can serve as both a mantissa multiplier and an accumulator. As illustrated in Figure 8, the adder-tree-based PE consists of multiple exponent units, an adder tree, and an accumulator. The operation mode of the adder-tree-based PE is determined by a control bit signal, which distinguishes between high-precision and low-precision operations. This bit signal is set by the importance mask.

When the bit signal is set to 0, the PE operates low-precision operations (as indicated by the blue operands in Figure 8). In this mode, the adder-tree-based PE performs multiple low-precision multiplications in parallel using several exponent units. As shown in Equation (1), low precision multiplication

is performed by adding the exponents of the input and weight and shifting the input mantissa accordingly. Thereby, each exponent unit receives a distinct pair of input and weight exponents, shifts the corresponding mantissa based on the summed exponent, and produces a multiplication result for that pair. The adder tree accumulates these results, completing the MAC operation across multiple input-weight pairs.

When the bit signal is set to 1, the PE operates in high-precision mode (as indicated by the red operands in Figure 8). In conventional floating-point computation, the mantissa unit generates partial products, which are then shifted according to the exponent. In contrast, when performing high-precision operations in the proposed adder-tree-based PE, this order is reversed: each exponent unit first produces pre-shifted partial products, and the adder tree subsequently accumulates them. To enable this process, all exponent units receive the same input and weight exponents, and each generates a partial product of either 0 or  $M_w$ , depending on the bit of the input mantissa. Summing all these pre-shifted partial products allows the PE to effectively perform standard floating-point multiplication.

Through this design, the PE operates as a conventional floating-point multiplier when high-precision computation is required. In contrast, during low-precision operation, it performs multiple low-precision MAC operations simultaneously, thereby the reduced workload of low-precision operations translate into a corresponding reduction in latency.

#### 4.2 Dataflow and Overall Scale-out Design

**Challenge** While the proposed adder-tree-based PE can efficiently handle both high-precision and low-precision MAC operations, its scalability when multiple PEs are deployed in parallel remains uncertain. Moreover, during the execution stage, high- and low-precision operations are often interleaved in irregular ratios without any distinct structural pattern, making it difficult to fit them into a predefined dataflow. To overcome these limitations, we propose an optimized dataflow that effectively reuses data.

**Proposed Execution Stage Dataflow** The proposed execution stage dataflow is based on two key observations: (1) a high-precision operation can be divided into a mantissa

multiplication and low-precision operations, and (2) our approach is insensitive to the execution order of high- and low-precision operations during the execution stage. Unlike conventional IDPS, which identifies important neurons based on the results of low-precision computation, our method determines important neurons during a separate prediction stage. Consequently, the high-precision operation in the execution stage can be divided into mantissa multiplication and low-precision operations (exponent computations), where the mantissa multiplication can be performed first. The remaining low-precision operations of important neurons are then executed and accumulated together with those of less important neurons, resulting in a more structured dataflow.

As illustrated in Figure 9, the proposed dataflow in the execution stage for each layer is divided into two phases: (1) mantissa multiplication and (2) low-precision computation. In the mantissa multiplication phase, the mantissa products required for computing important neurons in the corresponding layer are precomputed. The input rows and weight columns necessary for these computations are fetched, and the values of each row are delivered to the PEs through the on-chip network for PE initialization. Each PE performs only mantissa multiplication on the received values, without conducting exponent computation or accumulation. The resulting mantissa multiplication outputs are then transferred via the on-chip network and stored in the buffer. Since each access involves a single but wide row and column of data, the dataflow avoids irregular, small-granularity memory accesses. Furthermore, because accumulation is not performed in this phase, all PEs can operate fully in parallel, maximizing parallelism and computational efficiency.

In the low-precision computation phase, the dataflow follows the same pattern as a conventional systolic array, but executes multiple low-precision operations in parallel. Also, when a PE is required to perform a high-precision operation, it retrieves the precomputed mantissa through the on-chip network, allowing both high- and low-precision computations to be processed within a structured dataflow. Figure 9 shows the low-precision computation section, shown in light green, where each adder-tree PE contains two exponent units, enabling it to perform two pairs of input-weight MAC operations simultaneously. Consequently, weights propagate to the right and partial sum propagate downward as in Figure 6, except that two weights are transferred at a time. The low-precision computation operates as follows. Each PE initializes the exponents and mantissas of its assigned inputs via the on-chip network (①) and diagonally passes the weight exponents from left to right. (② ~④) During low-precision computation, each PE uses the locally stored input mantissa and exponent together with the weight exponents received from the PE above to perform the operation. When a high-precision computation is required, the PE instead uses the

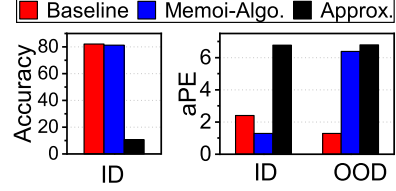


Figure 10: Algorithm.

| Pass # | Algo. | Hard. |
|--------|-------|-------|
| 25     | 1.08  | 0.405 |
| 50     | 1.04  | 0.365 |
| 100    | 1.02  | 0.142 |

Table 1: Hardware.

precomputed mantissa received from the on-chip network, along with the stored input exponent and the propagated weight exponent, to perform multiplication. In the Figure 9, the green-highlighted PEs indicate those performing high-precision operations, where the on-chip network is utilized to deliver the precomputed multiplied mantissas. This approach enables mixed-precision computation without losing the inherent data-reuse behavior of weights and partial sums, by leveraging the existing on-chip network.

## 5 Evaluation

### 5.1 Algorithm Evaluation

**Evaluation Setup** We evaluate the impact of the Memoization Algorithm on inference accuracy and uncertainty estimation using the ImageNet dataset [10] for Vision Transformer [8]. The out-of-distribution (OOD) images are generated using random noise. For the MCD-BNN evaluation, each experiment was repeated five times for both the floating-point baseline and the Memoization Algorithm to mitigate stochastic effect, with the dropout rate set to 0.4.

**Algorithm Evaluation** In Figure 10, the baseline represents the case where all operations are executed in high precision, while Memoi-Algo. corresponds to the case where only important neurons are computed in high precision. In contrast, Approx. denotes the configuration where all neurons are computed in low precision. The results show that Memoi-Algo. achieves almost the same inference accuracy as the baseline (less than a 1% accuracy drop). Furthermore, while both Approx. and Memoi-Algo. perform approximate computations with ratios of 100% and 80%, respectively, only Memoi-Algo. is able to produce correct inference results, whereas Approx. fails to do so.

The second graph of Figure 10 presents the average predictive entropy (aPE), which measures uncertainty estimation [11]. Ideally, the aPE should be higher for OOD datasets and lower for in-distribution (ID) datasets, indicating that the MCD BNN correctly captures predictive uncertainty. As shown, both the baseline and Memoi-Algo. exhibit a clear separation between ID and OOD conditions, demonstrating that our method preserves the uncertainty estimation capability inherent to Bayesian inference. In contrast, the Approx. configuration fails to distinguish between ID and

OOD datasets, indicating a degradation in its ability to model predictive uncertainty.

## 5.2 Accelerator Evaluation

**Evaluation Setup** We evaluate both the proposed Memoization Hardware and the baseline systolic array, each targeting the Bfloat16 format, to analyze their performance when executing the MCD-Bayesian ResNet-18 model. The ResNet-18 maintains its accuracy even when approximately 90% of computations are performed in low precision.

The Memoization Hardware is implemented with each adder-tree-based PE equipped with four exponent units, and the on-chip network bandwidth for initializing PE is extended to 28 bits, enabling low-precision operations to be executed up to four times faster. The systolic array configuration follows that of Eyeriss[23].

**Accelerator Evaluation** The Pass # column in Table 1 indicates the number of stochastic forward passes required for a single MCD-BNN inference. Algo. and Hard. represent the normalized cycle counts when executing the Memoization Algorithm on the conventional systolic array and on the proposed Memoization Hardware, respectively. Each value is normalized to the cycle count of the baseline MCD-BNN inference (where all stochastic forward passes are performed in floating-point precision), executed on the conventional systolic array. In the conventional systolic array, the reduced computation from low-precision operations in the Memoization Algorithm does not yield latency improvement but rather results in mantissa unit underutilization. As a result, Algo. performs worse than the baseline MCD-BNN, with only additional prediction overhead introduced. In contrast, the proposed Memoization Hardware, which employs adder-tree-based PEs and a tailored dataflow, effectively exploits the reduced computation introduced by the Memoization Algorithm. As a result, the total number of execution cycles is reduced to average 30% of that required by the baseline. Moreover, as the number of stochastic forward passes per inference increases, the performance advantage of the Memoization Hardware becomes more significant, because the prediction overhead is increasingly amortized across passes.

## 6 Conclusion

We designed a domain-specific accelerator that enables fast and efficient estimation of uncertainty, through an algorithm-architecture co-optimization that exploits the characteristics of MCD BNNs.

## 7 Discussion

In this study, the area overhead introduced by the modified processing element (PE) architecture and network-on-chip

was not evaluated. As part of future work, we plan to implement both the conventional floating-point PE and the proposed adder-tree-based PE in Verilog, followed by synthesis to measure their respective area and power consumption, as well as to obtain precise cycle counts required for model execution. In addition, we intend to extend our evaluation to a broader set of neural network models to further validate the generality of the proposed approach. Furthermore, we plan to investigate strategies for precomputed mantissas of important neurons, including how to efficiently store and manage them within the on-chip buffer.

## Acknowledgements

This work was supported by Samsung Research Funding & Incubation Center for Future Technology of Samsung Electronics under Project Number SRFC-IT2302-05.

## References

- [1] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.
- [2] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*, 2015.
- [3] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, 30, 2017.
- [4] Mingcong Song, Jiechen Zhao, Yang Hu, Jiaqi Zhang, and Tao Li. Prediction based execution on deep neural networks. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 752–763. IEEE, 2018.
- [5] Liqiang Lu, Yicheng Jin, Hangrui Bi, Zizhang Luo, Peng Li, Tao Wang, and Yun Liang. Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 977–991, 2021.
- [6] Liu Liu, Zheng Qu, Lei Deng, Fengbin Tu, Shuangchen Li, Xing Hu, Zhenyu Gu, Yufei Ding, and Yuan Xie. Duet: Boosting deep neural network efficiency on dual-module architecture. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 738–750. IEEE, 2020.
- [7] Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015.
- [8] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- [9] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [10] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.
- [11] Hongxiang Fan, Martin Ferianc, and Wayne Luk. Enabling fast uncertainty estimation: accelerating bayesian transformers via algorithmic and hardware optimizations. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 325–330, 2022.

- [12] Zehuan Zhang, Hongxiang Fan, Hao Chen, Lukasz Dudziak, and Wayne Luk. Hardware-aware neural dropout search for reliable uncertainty prediction on fpga. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6, 2024.
- [13] Hongxiang Fan, Martin Ferianc, Zhiqiang Que, Xinyu Niu, Miguel Rodrigues, and Wayne Luk. Accelerating bayesian neural networks via algorithmic and hardware optimizations. *IEEE Transactions on Parallel and Distributed Systems*, 33(12):3387–3399, 2022.
- [14] Hongxiang Fan, Martin Ferianc, Miguel Rodrigues, Hongyu Zhou, Xinyu Niu, and Wayne Luk. High-performance fpga-based accelerator for bayesian neural networks. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 1063–1068. IEEE, 2021.
- [15] Qiyu Wan and Xin Fu. Fast-bcnn: Massive neuron skipping in bayesian convolutional neural networks. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 229–240. IEEE, 2020.
- [16] Yeonhong Park, Jake Hyun, Hojoon Kim, and Jae W Lee. {DecDEC}: A systems approach to advancing {Low-Bit} {LLM} quantization. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, pages 803–819, 2025.
- [17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [18] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.
- [19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.
- [20] Samer Kurzum, Gil Shomron, Freddy Gabbay, and Uri Weiser. Enhancing dnn training efficiency via dynamic asymmetric architecture. *IEEE Computer Architecture Letters*, 22(1):49–52, 2023.
- [21] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E Haque, Lingjia Tang, and Jason Mars. The architectural implications of autonomous driving: Constraints and acceleration. In *Proceedings of the twenty-third international conference on architectural support for programming languages and operating systems*, pages 751–766, 2018.
- [22] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.
- [23] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1):127–138, 2016.
- [24] Yun-Chen Lo, Yu-Chih Tsai, and Ren-Shuo Liu. Lv: Latency-versatile floating-point engine for high-performance deep neural networks. *IEEE Computer Architecture Letters*, 22(2):125–128, 2023.
- [25] Jonathan Ying Fai Tong, David Nagle, and Rob A Rutenbar. Reducing power by optimizing the necessary precision/range of floating-point arithmetic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3):273–286, 2000.