Hardware-Aware Neural Dropout Search for Reliable Uncertainty Prediction on FPGA

Zehuan Zhang
Department of Computing
Imperial College London
zehuanzhang22@imperial.ac.uk

Hongxiang Fan*
Imperial College London &
Samsung AI Center
hongxiangfan@ieee.org

Mark Chen
Department of Computing
Imperial College London
hao.chen20@imperial.ac.uk

Lukasz Dudziak
Samsung AI Center
l.dudziak@samsung.com

Wayne Luk
Department of Computing
Imperial College London
w.luk@imperial.ac.uk

ABSTRACT

The increasing deployment of artificial intelligence (AI) for critical decision-making amplifies the necessity for trustworthy AI, where uncertainty estimation plays a pivotal role in ensuring trustworthiness. Dropout-based Bayesian Neural Networks (BayesNNs) are prominent in this field, offering reliable uncertainty estimates. Despite their effectiveness, existing dropout-based BayesNNs typically employ a uniform dropout design across different layers, leading to suboptimal performance. Moreover, as diverse applications require tailored dropout strategies for optimal performance, manually optimizing dropout configurations for various applications is both error-prone and labor-intensive. To address these challenges, this paper proposes a novel neural dropout search framework that automatically optimizes both the dropout-based BayesNNs and their hardware implementations on FPGA. We leverage one-shot supernet training with an evolutionary algorithm for efficient dropout optimization. A layer-wise dropout search space is introduced to enable the automatic design of dropout-based BayesNNs with heterogeneous dropout configurations. Extensive experiments demonstrate that our proposed framework can effectively find design configurations on the Pareto frontier. Compared to manually-designed dropout-based BayesNNs on GPU, our search approach produces FPGA designs that can achieve up to 33× higher energy efficiency. Compared to state-of-the-art FPGA designs of BayesNN, the solutions from our approach can achieve higher algorithmic performance and energy efficiency.

1 INTRODUCTION

Deep neural networks (DNNs) are pervasively utilized in various domains [4], achieving superior performance. Despite their capability, conventional DNNs are prone to overfitting, and thus are not able to indicate potential issues in their predictions. This may cause silent failures for safety-critical applications [22], compromising trustworthiness in deep learning for vital decision-making processes. The capability of providing uncertainty estimation is crucial in mitigating the inherent risks in deep learning, ensuring predictions come with well-calibrated confidence levels. Bayesian Neural Network [18] (BayesNN) emerges as a highly effective method for reliable uncertainty estimation. Various approximation techniques have been introduced [2, 16, 17] for BayesNN. Among these methods, dropout-based methods [13] have emerged as one of the mainstreaming approaches for reliable uncertainty estimation due to their compute and memory efficiency. Existing research has delved into exploring different dropout designs with diverse granularities and sampling dynamics [14, 21].

Although a substantial amount of progress has been made in this research field, there are still challenges in deploying dropout-based BayesNNs in real-life applications. First, existing dropout-based BayesNNs are predominantly designed with a single dropout layer type throughout the network. This may lead to sub-optimal performance as different stages of the network require specialized dropout designs with distinct dropout granularities and sampling approaches to optimize performance. Second, diverse applications necessitate tailored dropout strategies to optimize performance. It is a labor-intensive and heuristic-driven process to manually find the optimal dropout configurations for the target application. Third, the computational and memory requirements of dropout-based BayesNNs necessitate hardware acceleration in practical deployment scenarios. However, previous accelerators [1, 3, 7, 9–12] are only designed for dropout-based BayesNN with a uniform dropout strategy. Also, the hardware efficiency of different dropout strategies is not considered while designing the dropout-based BayesNN accelerators.

To address the above challenges, this paper proposes a novel neural dropout search framework that automatically optimizes both the dropout-based BayesNNs and the associated hardware accelerators on FPGA. The optimization of dropout strategies is formulated as a search problem with both algorithmic and hardware performance as the main objectives. We develop a layer-wise dropout search space, allowing the hybrid use of different dropout layers at different network stages. Our novel four-phase design framework automates the search of optimal dropout strategies, tailored to specific applications and constraints. The proposed framework leverages one-shot

*Corresponding author

DAC '24, June 23–27, 2024, San Francisco, CA, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0601-1/24/06
https://doi.org/10.1145/3649329.3656528

supernet training combined with an evolutionary algorithm to ensure efficient optimization. Moreover, we introduce FPGA-based implementations for four distinct dropout designs, enabling the FPGA-based hardware acceleration of dropout-based BayesNNs with heterogeneous dropout layers. Our code is publicly available at: https://github.com/zehuanzhang/Neural_Dropout_Search.git.

Our contributions can be summarized as follows:

- A novel neural dropout search framework with one-shot supernet training and an evolutionary algorithm to automatically optimize both dropout-based BayesNNs and the associated FPGA-based accelerators given the target applications and constraints.
- A layer-wise dropout search space that enables the automatic optimization of dropout-based BayesNNs with heterogeneous dropout layers for higher performance.
- FPGA-based implementations of four types of dropout designs, enabling the acceleration of dropout-based BayesNNs with different dropout combinations.

2 BACKGROUND AND RELATED WORK

2.1 Background

2.1.1 Bayesian Neural Network. In contrast to conventional DNNs, BayesNNs employ probabilistic inference to provide uncertainty estimates for predictions. A comprehensive literature review of BayesNNs is summarized in [17]. Instead of presenting weights using point-wise values, BayesNNs are trained to construct the posterior distributions on weights through Bayesian inference. Leveraging this approach, BayesNNs are able to deliver reliable uncertainty estimation, mitigating overfitting issues and enhancing robustness.

However, given the high dimensionality inherent in modern BayesNNs, the analytical inference of the posterior distribution associated with the model weights and the subsequent predictions is computationally prohibitive. To resolve this issue, several approximation methods have been introduced [2, 16, 19], with dropout-based approximations showing significant promise.

2.1.2 Dropout-based Approximation. Dropout-based approximations emerge as efficient approaches for approximating BayesNNs. To obtain uncertainty estimation, a dropout-based BayesNN runs the forward pass multiple times with the dropout enabled during inference. Different dropout masks are generated in distinct forward passes to generate different Monte Carlo samples.

Employing dropout in both the training and inference stages of DNNs has demonstrated to perform Bayesian inference [13]. Based on this theoretical grounding, extensive studies have investigated various dropout designs with different granularities and sampling dynamics [14, 21]. As shown in Figure 1, this paper focuses on the four most representative dropout designs: Bernoulli [13], Block Dropout [14], Random Dropout and Masksembles [5].

2.2 Related Work

Given the substantial computational and memory demands, there has been significant research dedicated to accelerating BayesNNs. *VIBNN* [3] accelerated BayesNNs based on Variational inference with two Gaussian (pseudo) random number generators employed for efficient inference. *BYNQNet* [1] explored the sampling-free method with an FPGA-based implementation on a Xilinx PYNQ-Z1 board. *ASBNN* [12] examined the relationship among multiple forward passes for approximate calculations. To achieve higher hardware performance, multiple studies [8, 9, 11] explored structured sparsity in different granularities. [7] combined dropout methods and Multi-Exit networks to optimize dropout-based BayesNN with temporal and spatial mapping strategies.

These prior studies, however, predominantly involve manual designs of BayesNNs. Moreover, most approaches employing dropout-based BayesNNs neglect the vast potential offered by heterogeneous dropout layers. In contrast to prior studies, this paper systematically analyzes different dropout strategies. We propose a neural dropout search framework to find the optimal layer-wise dropout design, which is capable of automatically generating an FPGA-based BayesNNs accelerator for the target application.

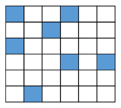
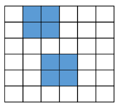
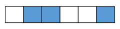
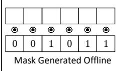
Name	RandomDropout	BlockDropout	BernoulliDropout	Masksembles
Granularity	Point	Patch	Point/Channel	Point/Channel
Dynamic	Dynamical	Dynamical	Dynamical	Static
Layer	CONV	CONV	FC/CONV	FC/CONV
Example				

Figure 1: Four dropout layers with different granularities and sampling dynamics.

3 NEURAL DROPOUT SEARCH

3.1 Framework Overview

Given specifications and search objectives, our framework aims to find the optimal dropout configurations. We formulate the optimization problem as follows:

$$\arg\min_{c\in C} L(c) \tag{1}$$

In the above, C denotes the dropout search space, and c denotes an individual dropout configuration. The overall training goal is to find the optimal dropout configuration scheme to meet algorithm and hardware requirements.

To address the optimization problem (1) above, we propose a novel neural dropout search framework as shown in Figure 2. To make the framework applicable to mainstream neural networks and dropout strategies, we systematically arrange it into four phases: (1) Specification, (2) Training, (3) Search, (4) Accelerator Generation.

3.2 Specification: Phase1

The framework starts by receiving the network architecture, heterogeneous dropout methods, and specified dropout layer positions as inputs. Typically, DNN architectures contain built-in dropout layers for regularization. Our framework is designed to support a wide variety of such DNNs. There are no restrictions imposed on network architectures or dropout strategies, so the proposed framework can support a wide spectrum of scenarios.

A portion of built-in dropout layers is specified. Each specified dropout layer is assigned with multiple heterogeneous dropout methods, for constructing a supernet. The number of such specified dropout layers is denoted as N, and the number of dropout methods

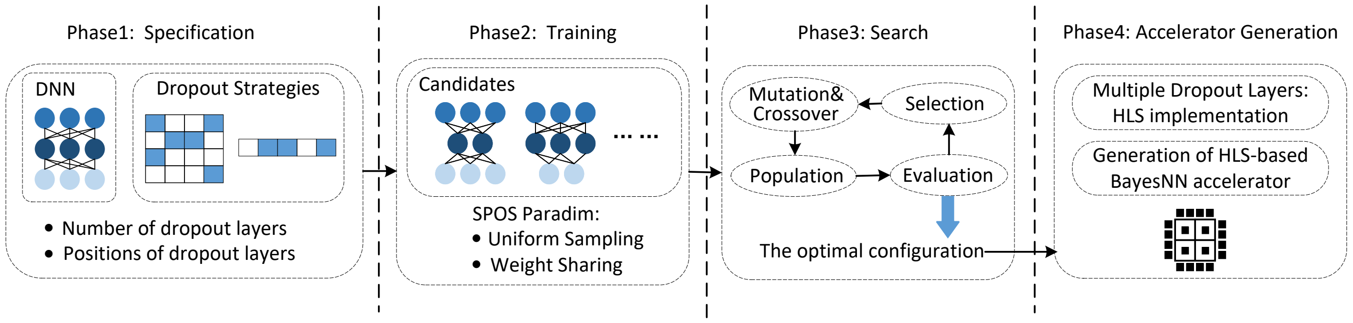


Figure 2: An overview of the proposed framework. HLS stands for High Level Synthesis.

in each is denoted as $M_i (i = 1, 2, ..., N)$. For each forward pass, each specified dropout layer selects a dropout method randomly. Consequently, the constructed supernet encompasses an ensemble of network architectures, amounting to a total of $\prod_{i=1}^{N} M_i$ sub-networks characterized by diverse configurations, each serving as a candidate within the total search space. Note that candidates with uniform and hybrid dropout configurations are both allowed considerably expanding the exploration space.

Regarding the heterogeneous dropout methods, four dropout designs—Random Dropout, Block Dropout [14], Bernoulli Dropout [13] and Masksembles [5]—are employed. The first three dropout methods support dynamic adoption with different levels of granularity, while the last one is static and can be placed following both convolutional layers and fully connected layers. Different dropout designs offer a means to strike a balance between accuracy, uncertainty estimation, and hardware performance.

3.3 Training: Phase2

As stated in Eq (1), the training process can be cast as a multi-objective problem. A naive approach to solve this is to directly train the supernet from scratch using the gradients of the overall objective derived by evaluating all sub-networks at each update step. However, this approach incurs an increase in training costs proportional to the number of sub-networks. If the size of the search space is enormous, it is challenging to train every candidate sub-network. Moreover, when handling different tasks, it necessitates retraining the supernet, which can be computationally prohibitive.

Inspired by [15], the Single Path One-Shot (SPOS) paradigm is exploited for supernet training. Within each training iteration, a candidate sub-network is uniformly sampled by randomly selecting the dropout strategy in each specified layer. This paradigm allows for the sharing of weights across all sub-networks, thus enabling convergence of each candidate sub-network even if the search space is enormous. As such, it results in substantial reduction of the overall training costs from $O(\prod_{i=1}^{N} M_i)$ to $O(1)$, significantly addressing the computational challenges posed. Furthermore, the training and searching stages are decoupled. The supernet needs to be trained only once. Subsequently, each candidate sub-network can be directly sampled with shared weights in the search phase.

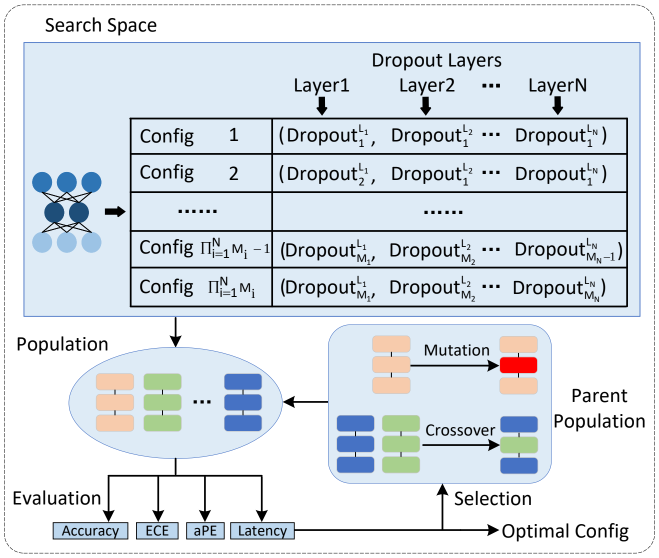

Search Space
Dropout Layers
Layer1
Layer2
LayerN
Config 1
Config 2
Config $\prod_{i=1}^{N} M_i - 1$
Config $\prod_{i=1}^{N} M_i$
(Dropout$_1^{L_1}$, Dropout$_1^{L_2}$ ··· Dropout$_1^{L_N}$)
(Dropout$_2^{L_1}$, Dropout$_1^{L_2}$ ··· Dropout$_1^{L_N}$)
(Dropout$_{M_1}^{L_1}$, Dropout$_{M_2}^{L_2}$ ··· Dropout$_{M_N-1}^{L_N}$)
(Dropout$_{M_1}^{L_1}$, Dropout$_{M_2}^{L_2}$ ··· Dropout$_{M_N}^{L_N}$)
Population
Mutation
Crossover
Parent Population
Evaluation
Selection
Accuracy
ECE
aPE
Latency
Optimal Config

Figure 3: Evolutionary Algorithm.

3.4 Search: Phase3

After training, the third phase is to conduct a thorough search for the optimal network configuration given the defined search aim, which takes into account both software and hardware performance. An evolutionary algorithm is adopted to search for the optimal configuration as shown in Figure 3.

Our evolutionary algorithm consists of four stages: population, evolution, selection, and crossover&mutation. First, the population stage initiates by randomly sampling a set of candidate sub-networks with distinct dropout configurations. Each candidate sub-network can be represented as a combination of dropout designs. The sampling process continues until the candidate pool reaches the predefined size. Second, the performance of all the sampled candidates in the population is evaluated on the validation set. Third, the top-performing solutions are selected as the parent population. Fourth, a fraction of candidates in the parent population undergo mutations, while the others undergo crossovers. During the mutation process, each candidate experiences random mutation in the dropout types with a predefined probability. During the crossover process, a pair of candidates are selected randomly. A new configuration is produced by swapping each dropout type randomly within the specified dropout layers. Through the crossover and mutation processes, new network configurations are generated to update the population candidates. Such sampling and updating processes are repeated until the optimal configuration is found.

Within our framework, we choose the accuracy metric to assess the predicted performance, the expected calibration error (ECE) and average prediction entropy (aPE) metrics to assess the uncertainty performance, and hardware latency to measure the running speed for each potential candidate. The algorithmic metrics are evaluated on the validation set. The search aim is expressed as:

$$aim = \eta \times Accuracy - \mu \times ECE + \beta \times aPE - \lambda \times Latency \quad (2)$$

Here η, μ, β and λ represent the weights of the respective metrics in the search. As the lower the ECE and latency are, the higher the overall performance is, so these two metrics are negative terms. The final goal is to find a network architecture that maximizes both accuracy and uncertainty performance while minimizing hardware latency. As the metrics cover trade-offs that need to be considered and balanced, it is allowed to prioritize these metrics and establish the search aim accordingly tailored to specific requirements.

3.5 Accelerator Generation: Phase4

3.5.1 Performance modelling. As explained above, the third phase of our framework employs an evolutionary algorithm to optimize both algorithmic and hardware settings. Each iteration of the evolutionary algorithm necessitates running the hardware implementation to assess the performance of the chosen candidates for subsequent optimization. Given that design synthesis and place & route operations in FPGA hardware implementation are time-consuming and often need manual intervention, the optimization process can become lengthy. To expedite this process, we introduce a machine learning-based hardware cost model, facilitating rapid evaluation.

For accurate and fast estimation, we construct a training dataset that consists of multiple data points, where hardware configurations serve as inputs and predicted values of latency as the output. The hardware configurations include the input shape and dropout type. We employ Gaussian process [20] for regression using this dataset. It is worth noting that both dataset construction and training are only required once. The Gaussian process model can be reused for various evolutionary optimizations with different search objectives. We choose Matérn kernel and constant mean function.

3.5.2 HLS-based implementation. The last phase generates the corresponding hardware accelerator using HLS (High Level Synthesis). The HLS4ML[6] open-source framework is adopted to generate our BayesNNs accelerators. There are existing HLS templates of normal layers in HL4ML to construct traditional DNNs. To support BayesNN accelerators, we introduce HLS-based implementation of the newly introduced dropout layers into the design flow. So hardware accelerators, designed using HLS, can then be integrated into Vivado-HLS for synthesis and place & route.

4 EXPERIMENTS

Experiments are implemented in Python 3.9, Pytorch 1.10.0, and Keras 2.9.0. The Intel Core i9-9900K CPU and one Nvidia GeForce GTX 2080 TI GPU are equipped in our machine. We use Vivado-HLS 2020.1 for hardware implementation. 16-bit fixed data is used, with 1 sign bit, 7 integer bits and 8 fraction bits. QKeras is used for quantization. The latency, resource utilization and power consumption are obtained from C-synthesis reports provided by Vivado-HLS. Vivado 2020.1 is used to run place and route for the final designs. We select Xilinx Kintex XCKU115 as our target FPGA board.

4.1 Effectiveness of Neural Dropout Search

To demonstrate the advantages of auto-generated configurations of dropout-based BayesNNs, we evaluate three commonly-used models, LeNet, VGG11 and ResNet18 for image classification on different datasets, MNIST, SVHN and Cifar-10, respectively. Synthetic data generated by random Gaussian noise with mean and standard deviation of the training data, are employed to measure aPE. There are four dropout choices: Bernoulli Dropout, Random Dropout, Block Dropout and Masksembles. For LeNet, we specified three dropout layers: (a) two dropout layers follow convolutional layers with all four dropout choices, (b) one dropout layer follows fully-connected layers with two dropout choices: Bernoulli Dropout and Masksembles. For VGG11 and ResNet18, we specify four dropout layers following convolutional layers with four dropout choices. The sampling number is set as three.

The results of ResNet18 on the test set of Cifar-10 are presented in Table 1. We adopt accuracy, ECE, aPE and latency as search aims, respectively. Table 1 shows that all the optimal configurations can be found. To further demonstrate the effectiveness of our search approach, we adjust search aims to find different Pareto-optimal designs. The Pareto-optimal points outperform any other points in either accuracy or uncertainty. As discussed, the weight parameters in the search aim represent the importance of different metrics. We iterate through and evaluate all configurations on the validation sets. To compare in a straightforward way, we highlight baselines configured with uniform dropout designs and the configurations obtained from search. As shown in Figure 4, all the searched results lie on the reference Pareto frontier, confirming that our framework can effectively identify optimal designs. Therefore, for real applications, multiple configurations can be obtained depending on different aims, providing strong flexibility.

The search costs and the resulting configurations are shown in Table 2. To achieve the highest accuracy, the optimal dropout configurations for LeNet, VGG11, and ResNet18 are all hybrid dropout configurations. In previous work, the networks are typically designed with uniform dropout configurations manually. Hybrid dropout configurations are rarely designed by human experts, leading to oversight of optimal designs. Moreover, in some cases, the uniform configurations may achieve performance even below the average level, such as the 'All Random Dropout' in ResNet in Figure 4. This further emphasizes the benefits of our approach. Given a vast design space and the variability of tasks, our auto-search framework can effectively uncover configurations that may be non-intuitive to human experts yet yield superior performance outcomes.

4.2 Comparison with CPU and GPU

We compare our approach against CPU and GPU implementations as shown in Table 3. The comparison uses MNIST dataset since

Table 1: Algorithm and hardware results of optimized configurations obtained from search.

ResNet	Configurations	Accuracy (%)↑	ECE (%)↓	aPE (nats)↑	Latency (ms)↓	Resource Utilization		
						BRAM	DSP	FF
Uniform Configurations	All Bernoulli Dropout	91.205	7.4	0.989	15.401	82%	5%	40%
	All Block Dropout	91.276	5.9	0.887	18.674	82%	5%	39%
	All Random Dropout	90.635	5.8	0.773	18.396	82%	5%	39%
	All Masksembles	91.316	3.6	0.626	15.401	82%	5%	39%
Searched Configurations	Accuracy Optimal	**91.456**	5.5	0.784	18.671	82%	5%	40%
	ECE Optimal	91.316	**3.6**	0.626	15.401	82%	5%	39%
	aPE Optimal	91.205	7.4	**0.989**	15.401	82%	5%	40%
	Latency Optimal	91.206	7.4	0.626	**15.401**	82%	5%	39%

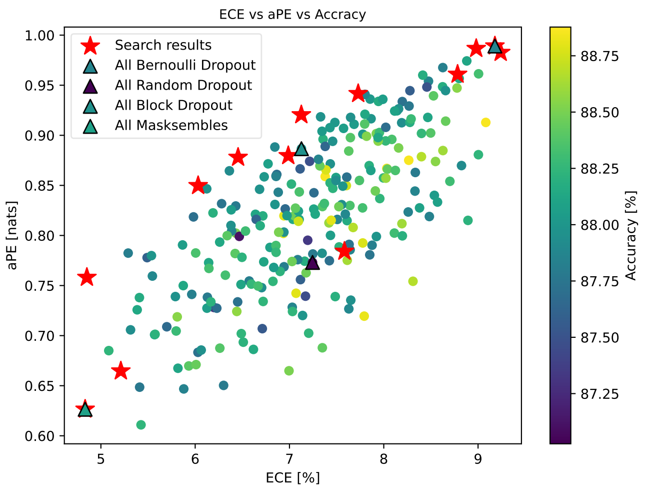


Figure 4: Search results.

Table 2: Search costs and resultant configurations on three networks. (B: Bernoulli Dropout, R: Random Dropout, K: Block Dropout, M: Masksembles)

	Search Cost (GPU)	Dropout Configurations
LeNet	~2 hours	Accuracy Optimal: B - B - M
		ECE Optimal: M - M - B
		aPE Optimal: R - R - B
		Latency Optimal: M- M- M
VGG	~6 hours	Accuracy Optimal: R - B - B - R
		ECE Optimal: R - K - R - M
		aPE Optimal: R - R - R - R
		Latency Optimal: M- M- M -M
ResNet	~10 hours	Accuracy Optimal: K - M - B - M
		ECE Optimal: M - M - M - M
		aPE Optimal: B - B - B - B
		Latency Optimal: M- M- M -M

it is the most common dataset in prior work. For dropout-based BayesNNs, the Bernoulli Dropout method is adopted as it is widely employed in hand-crafted approaches. We implement the auto-search configurations targeting the aPE Optimal design.

Regarding aPE performance, our auto-search design outperforms the model configured with a uniform Bernoulli Dropout. In terms of hardware performance, our design achieves 1.4 times speedup over CPU implementation. The power consumption of our design is 52.6× and 60.5× lower than that of CPU and GPU implementations. Consequently, our design achieves 65× and 33× higher energy efficiency over CPU and GPU implementations, despite the more advanced 14nm technology used in CPU and 12nm in GPU.

4.3 Comparison with Related Work

To compare with related work, we quote results from relevant papers. The results are shown in Table 3. Both [3] and [1] do not support LeNet. In terms of hardware performance, our accelerator achieves 6.1 times and 5.0 times faster speed than [3] and [1], and shows 8.2 times and 3.0 times higher energy efficiency than those, respectively. To compare with the work in [10], we reproduce the experiments using its techniques with the same sampling number as ours to report the aPE metric. The hardware performance is quoted from [10]. Our design achieves better aPE than [10], demonstrating the advantages of our auto-search framework. Although it operates at a lower latency, it incurs higher power costs, resulting in lower energy efficiency than our work. Since the design in [7] adopts multi-exit optimization to enhance performance, we are not able to compare against it. As their techniques are orthogonal to ours, we intend to incorporate these optimizations in future.

Figure 5 presents the power consumption breakdown of Accuracy Optimal and ECE Optimal designs, estimated from the Vivado tool after place and route. The Logic&Signal accounts for 39% and 32% of the dynamic power, respectively. The high consumption is due to the comparing operations in dynamic dropout layers. The BRAM accounts for 11% and 12% of the dynamic power. The implementation of Masksembles consumes more BRAM resources.

In addition, our design is more versatile and flexible than existing designs. The accelerators [1, 3] only support BayesNNs consisting of fully-connected layers. Our design is not specific to a particular network, and more dropout types are accepted. The framework can also be extended to cover more deep learning architectures, providing high generality. Furthermore, our accelerator is designed using HLS, which offers the benefits of improved productivity and easy code portability compared with RTL design [10].

Table 3: Comparisons with related work.

-	CPU	GPU	ASPLOS'18 [3]	DATE'20 [1]	TPDS'22 [10]	Our Work
Platform	Intel Core i9-9900K	NVIDIA RTX 2080	Altera Cyclone V	Zynq XC7Z020	Arria 10 GX1150	XCKU115
Frequency(MHz)	3600	1545	213	200	220	181
Technology	14 nm	12 nm	28nm	28nm	20nm	20nm
Power(W)	205	236	6.11	2.76	43.6	3.9
aPE(nats)↑	0.27	0.27	-	-	0.45	**0.65**
Latency(ms)	1.26	0.57	5.5	4.5	0.32	0.905
Energy Efficiency(J/Image)↓	0.258	0.134	0.033	0.012	0.014	**0.004**

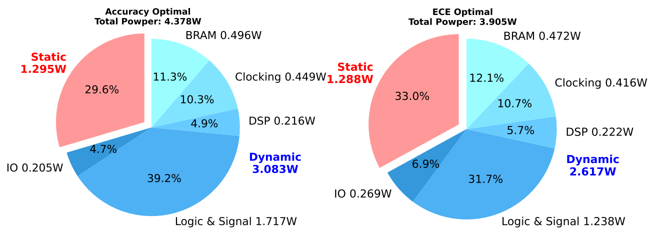


Figure 5: Power breakdown of the search designs.

5 CONCLUSION

This paper proposes a novel neural dropout search framework for automatic optimization of dropout-based BayesNNs and their implementation as hardware accelerators targeting FPGA technology. We leverage one-shot supernet training and an evolutionary algorithm approach to search for the optimal dropout designs. A layer-wise dropout search space is introduced to enable hybrid dropout design for BayesNNs. We propose hardware implementation for four different dropout layers, allowing the efficient mapping of the optimized BayesNNs onto an FPGA device.

Extensive experiments demonstrate that our approach can effectively identify the Pareto frontier designs with higher performance than prior methods. Further research includes incorporating additional dropout designs into our search space, providing sparsity support for hardware design, and extending the proposed framework to cover other kinds of neural networks such as Transformer and Graph Neural Network.

ACKNOWLEDGEMENT

The support of UK EPSRC (grant number EP/X036006/1, EP/P010040/1, EP/V028251/1 and EP/S030069/1) is gratefully acknowledged.

REFERENCES

[1] Hiromitsu Awano and Masanori Hashimoto. 2020. BYNQNet: Bayesian neural network with quadratic activations for sampling-free uncertainty estimation on FPGA. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1402–1407.

[2] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. 2017. Variational inference: A review for statisticians. *Journal of the American statistical Association* 112, 518 (2017), 859–877.

[3] 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. *ACM SIGPLAN Notices* 53, 2 (2018), 476–488.

[4] Shi Dong, Ping Wang, and Khushnood Abbas. 2021. A survey on deep learning and its applications. *Computer Science Review* 40 (2021), 100379.

[5] Nikita Durasov, Timur Bagautdinov, Pierre Baque, and Pascal Fua. 2021. Masksembles for uncertainty estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 13539–13548.

[6] Farah Fahim, Benjamin Hawks, Christian Herwig, James Hirschauer, Sergo Jindariani, Nhan Tran, Luca P Carloni, Giuseppe Di Guglielmo, Philip Harris, Jeffrey Krupa, et al. 2021. hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices. *arXiv preprint arXiv:2103.05579* (2021).

[7] Hongxiang Fan, Mark Chen, Liam Castelli, Zhiqiang Que, He Li, Kenneth Long, and Wayne Luk. 2023. When Monte-Carlo Dropout meets multi-exit: Optimizing Bayesian neural networks on FPGA. In *ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[8] Hongxiang Fan, Martin Ferianc, and Wayne Luk. 2022. Enabling fast uncertainty estimation: accelerating bayesian transformers via algorithmic and hardware optimizations. In *ACM/IEEE Design Automation Conference (DAC)*. 325–330.

[9] Hongxiang Fan, Martin Ferianc, Zhiqiang Que, Shuanglong Liu, Xinyu Niu, Miguel RD Rodrigues, and Wayne Luk. 2022. FPGA-based acceleration for Bayesian convolutional neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 12 (2022), 5343–5356.

[10] Hongxiang Fan, Martin Ferianc, Zhiqiang Que, Xinyu Niu, Miguel Rodrigues, and Wayne Luk. 2022. Accelerating bayesian neural networks via algorithmic and hardware optimizations. *IEEE Transactions on Parallel and Distributed Systems* 33, 12 (2022), 3387–3399.

[11] Hongxiang Fan, Martin Ferianc, Miguel Rodrigues, Hongyu Zhou, Xinyu Niu, and Wayne Luk. 2021. High-performance FPGA-based accelerator for Bayesian neural networks. In *ACM/IEEE Design Automation Conference (DAC)*. 1063–1068.

[12] Yoshiki Fujiwara and Shinya Takamaeda-Yamazaki. 2021. ASBNN: Acceleration of Bayesian Convolutional Neural Networks by Algorithm-hardware Co-design. In *International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. 226–233.

[13] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning (ICML)*. 1050–1059.

[14] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. 2018. Dropblock: A regularization method for convolutional networks. In *Advances in Neural Information Processing Systems (NeurIPS)*. 10727–10737.

[15] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2020. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision (ECCV)*. 544–560.

[16] W Keith Hastings. 1970. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* 57, 1 (1970), 97–109.

[17] 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.

[18] Radford Neal. 1992. Bayesian learning via stochastic dynamics. In *Advances in Neural Information Processing Systems (NeurIPS)*. 475–482.

[19] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, David Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. 2019. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 32.

[20] Christopher KI Williams and Carl Edward Rasmussen. 2006. *Gaussian processes for machine learning*. Vol. 2. MIT press Cambridge.

[21] Zhilu Zhang, Adrian V Dalca, and Mert R Sabuncu. 2019. Confidence calibration for convolutional neural networks using structured dropout. *arXiv preprint arXiv:1906.09551* (2019).

[22] Ke Zou, Zhihao Chen, Xuedong Yuan, Xiaojing Shen, Meng Wang, and Huazhu Fu. 2023. A Review of Uncertainty Estimation and its Application in Medical Imaging. *arXiv preprint arXiv:2302.08119* (2023).