Enabling Fast Uncertainty Estimation: Accelerating Bayesian Transformers via Algorithmic and Hardware Optimizations

Hongxiang Fan
Department of Computing
Imperial College London
London, UK
h.fan17@imperial.ac.uk

Martin Ferianc
Department of Electronic and Electrical Engineering
University College London
London, UK
martin.ferianc.19@ucl.ac.uk

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

ABSTRACT

Quantifying the uncertainty of neural networks (NNs) has been required by many safety-critical applications such as autonomous driving or medical diagnosis. Recently, Bayesian transformers have demonstrated their capabilities in providing high-quality uncertainty estimates paired with excellent accuracy. However, their real-time deployment is limited by the compute-intensive attention mechanism that is core to the transformer architecture, and the repeated Monte Carlo sampling to quantify the predictive uncertainty. To address these limitations, this paper accelerates Bayesian transformers via both algorithmic and hardware optimizations. On the algorithmic level, an evolutionary algorithm (EA)-based framework is proposed to exploit the sparsity in Bayesian transformers and ease their computational workload. On the hardware level, we demonstrate that the sparsity brings hardware performance improvement on our optimized CPU and GPU implementations. An adaptable hardware architecture is also proposed to accelerate Bayesian transformers on an FPGA. Extensive experiments demonstrate that the EA-based framework, together with hardware optimizations, reduce the latency of Bayesian transformers by up to 13, 12 and 20 times on CPU, GPU and FPGA platforms respectively, while achieving higher algorithmic performance.

KEYWORDS

Bayesian transformer, Adaptable micro-architecture, Sparsity

1 INTRODUCTION

Transformers [26] have become the leading approach in a wide range of natural language processing (NLP) [4] or computer vision (CV) tasks [31] by virtue of the attention mechanism. However, due to their deterministic inference, standard transformers cannot reliably quantify their uncertainty on predictions. Invalid uncertainty estimates can result in overconfident and uncalibrated decisions, which present hazards for deploying NNs in safety-critical applications such as in healthcare or autonomous driving [11, 15]. To overcome this drawback, Bayesian transformers [10, 22, 32] have been introduced with the mathematical grounding for reliable uncertainty estimation. An illustrative example is presented in Figure 1. Given a valid and an invalid sentence, the Bayesian transformer will be rightfully certain and correct on the valid sentence, while being uncertain given a random sentence. In practice, this uncertainty can be recorded and used to judge whether the system is certain enough to make a decision or additional expertise is required.

Among various Bayesian transformers, Monte Carlo Dropout (MCD)-based transformers have become the mainstream approach for providing reliable uncertainty estimation [8, 22]. However, the repeated Monte Carlo (MC) sampling and the compute-intensive attention mechanism deteriorate their hardware performance, limiting their deployment in real-world applications. For instance, our experiments show that a Bayesian RoBERTa [13] inferred though MCD with 20 MC samples for a sentence sequence takes nearly 800 ms on an Intel Xeon Gold 6154 CPU to make a prediction paired with uncertainty estimation, which cannot meet the requirements of real-world applications, such as in autonomous driving [17].

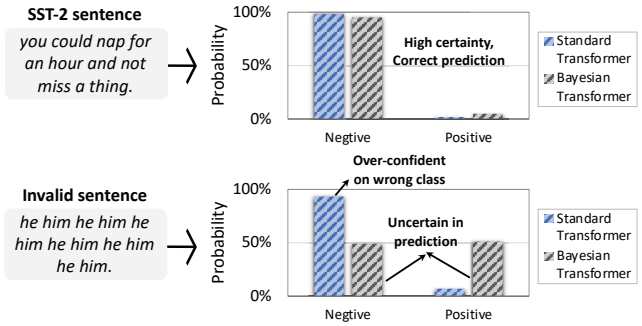


Figure 1: Comparison of standard and Bayesian transformers.

To address the aforementioned performance bottlenecks, we explore and exploit sparsity in Bayesian transformers to improve their hardware performance with respect to any hardware platform. According to their characteristics, this paper classifies the sparsity into three categories, namely head, block and sample sparsity. The three sparsity categories are controlled by dropout rates, the number of Bayesian blocks and the number of samples respectively. A higher sparsity represents a trade-off between hardware and algorithmic performance. To explore this trade-off, we propose an evolutionary algorithm (EA)-based framework to optimize the configurations of Bayesian transformers. On the hardware level, we show that the sparsity can be exploited to improve the hardware performance on our optimized CPU and GPU implementations. To further exploit the three sparsity categories on hardware, a highly optimized hardware architecture is proposed with the support of head, block and sample skipping. It also adopts a run-time adaptable computational engine to achieve high hardware performance.

Therefore, our contributions in this paper can be summarized as follows:

- A general framework to exploit the sparsity in Bayesian transformers, which improves their hardware performance on different hardware platforms without sacrificing their algorithmic performance (Section 3).

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530451

- A hardware architecture design for accelerating Bayesian transformers, which adopts a run-time adaptable computational engine, and exploitation of sparsity to achieve high hardware performance (Section 4).
- A comprehensive evaluation of various transformers on different datasets, which demonstrates improvements in both algorithmic and hardware performance on CPU, GPU and FPGA platforms (Section 5).

2 BACKGROUND AND RELATED WORK

2.1 Bayesian Transformer

2.1.1 Transformer Architecture. Based on the attention mechanism [26], transformers have been dominant in various NLP applications such as language modelling [18] and sentiment classification [4, 19]. The basic building module of transformers is the attention block, which mainly consists of a multi-head self-attention layer followed by a feed-forward network (FFN) [26]. According to the network structure, there are three categories of transformers, namely: encoder-only, decoder-only and encoder-decoder [19]. In this paper, our target application is sentence classification [28], hence, we mainly focus on accelerating and optimizing encoder-only transformers, such as BERT [4] and RoBERTa [13].

2.1.2 Bayesian Inference. Although transformers show excellent capability in processing long sequences of data, one of their main drawbacks is that they are not able to provide mathematically-grounded estimates of their uncertainty for predictions. To address this issue, Bayesian transformers have been proposed [22, 25, 32] with the ability to quantify their uncertainty. Among various Bayesian approaches, Monte Carlo Dropout (MCD) [8] has become a wide-spread Bayesian inference scheme [6, 22, 25, 27]. The concept of MCD lays in casting dropout [24], which is used both during training and evaluation, as Bayesian inference. As dropout is applied, it randomly sets certain connections as zeros. The dropout mask, which determines which connections are enabled or disabled, is generated by sampling from a Bernoulli distribution with a probability $p_{mcd} \in (0, 1)$. The prediction, along with uncertainty estimation, is obtained by performing MC sampling of the weights from the learnt posterior distribution and running the input through the network for the different sets of masks.

Dropout is naturally present in the transformer architecture, following the output of FFN [26]. Moreover, inspired by the drophead regularization [34], this paper applies dropout after the multi-head attention layer, which randomly drops out entire head outputs in the attention block. Dropping of heads assumes a variational distribution over the whole head weights instead of the individual weights as for FFN [33]. Note that, not every attention module needs to have dropout applied or have the same dropout rate p_{mcd}.

2.2 Hardware Accelerators

2.2.1 Transformer Accelerators. Various hardware architectures have been proposed to accelerate transformers, but they overwhelmingly focus on accelerating standard transformers. A^3 [9] and *SpAtten* [29] achieved 100× and 160× higher throughput than TITAN Xp GPU respectively. However, these accelerators did not optimize the FFN part, which limits their end-to-end hardware performance. *FTRANS* [12] attempted to accelerate transformer as a whole by optimizing pipeline strategies between different layers. Lu *et al.* [14] accelerated a single attention block based on a systolic array. Nevertheless, none of these designs investigated or accelerated the computation of Bayesian transformers.

2.2.2 Bayesian Neural Network Accelerators. Several hardware and algorithmic strategies have been introduced to accelerate Bayesian NNs [20]. Cai *et al.* [2] accelerated Bayesian multi-layer perceptrons (MLPs) on an FPGA. Awano & Hashimoto [1] improved the performance of Bayesian MLPs by adopting a sampling-free approach for uncertainty estimation. Targeting Bayesian convolutional NNs, [6] introduced FPGA-based accelerators with hardware and algorithmic optimizations for high hardware efficiency. *Fast-BCNN* [27] skipped the zeros caused by the element-wise ReLU activation for more efficient computation. However, these optimizations were designed for Bayesian convolutional or MLP nets, which cannot be straightforwardly applied to Bayesian transformers with different network structures and basic building blocks.

In comparison to the related work, our work thoroughly investigates the computation pattern in Bayesian transformers and identifies three different categories of sparsity. This sparsity is exploited through a novel optimization framework, optimized CPU and GPU implementations and a specialized FPGA-based hardware architecture for accelerating Bayesian transformers.

3 OPTIMIZATION FRAMEWORK

3.1 Sparsity Exploitation

To improve the hardware performance of Bayesian transformers, we analyse their compute pattern and identify extensive amount of sparsity in their computation that can be exploited. In this paper, we classify the sparsity into three categories: *i)* head sparsity, *ii)* block sparsity and *iii)* sample sparsity, which are illustrated in Figure 2.

The head sparsity is induced through employing MCD, where several heads in the Bayesian attention block are randomly dropped out with a dropout probability p_{MCD}. With MCD applied, the computation associated with the dropped heads is redundant, and thus can be skipped to achieve higher hardware performance. For instance, the forth head in the second Bayesian attention block of Figure 2 is dropped after MCD, so it is only needed to compute the first three heads to compute the output. The block sparsity is caused by the non-Bayesian attention blocks in Bayesian transformers. As the first attention block is non-Bayesian, we only need to compute it once in the first sample. Then, we can skip the first block while

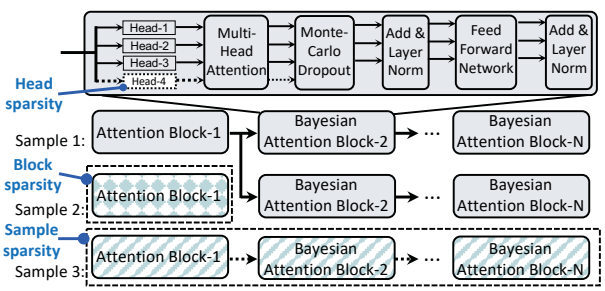


Figure 2: Sparsity categories in Bayesian Transformers.

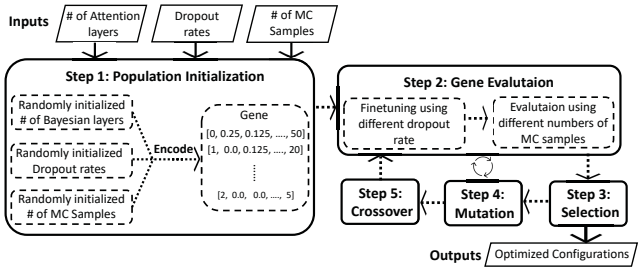


Figure 3: An overview of EA-based framework.

running the second sample because its results have been cached after the first sample. Lastly, we note the sample sparsity, which stands for redundant MC samples associated with the uncertainty estimation. Our experiments reveal that fewer samples can lead to the similar quality of uncertainty. In this paper, we systematically exploit these three categories of sparsity by considering both algorithmic and hardware performance of Bayesian transformers.

3.2 EA-based Optimization Framework

The head, block and sample sparsity decide the trade-off between the algorithmic and hardware performance, which is controlled by the dropout rates, the number of Bayesian layers and the number of samples respectively. However, these three parameters form a large design space, making it challenging to explore it efficiently. For instance, choosing the dropout rates from $\{0.1, 0.2, 0.4\}$ and the number of MC samples from $\{2, 5, 10, 20\}$ for a 24-layer Bayesian BERT-Large creates in total $3^{24} \times 4 = 1,129,718,145,924$ different configurations in the design space, which makes the optimization process time-consuming. To address this challenge, this work proposes an EA-based framework to optimize the configurations of Bayesian transformers with respect to algorithmic as well as hardware performance.

An overview of our proposed framework is presented in Figure 3. The framework accepts the baseline Bayesian transformer architecture as an input, including the combinations for the potential number of Bayesian attention blocks, available dropout rates and the executable number of MC samples. Then, the framework moves to the optimization stage. During the optimization, there are in total five steps: population initialization, evaluation, selection, mutation and crossover. In population initialization, we initialize N_{pop} different Bayesian transformers using random configurations. Then, different random configurations are encoded as genes for numerical optimization. We use the first and last elements of each gene to denote the numbers of Bayesian attention blocks and the number of MC samples respectively, while the rest of the gene is used to represent the dropout rates for each Bayesian attention block. Prior to the evaluation step, we finetune the pre-trained transformer using the dropout rates specified by the current genes, replacing the original dropout rates [22]. The finetuned transformers are then evaluated using different number of samples to obtain their algorithmic or hardware performance. Given their performance, the selection step chooses the best $N_{parents}$ genes as the parents for the next iteration, and the mutation is then applied to randomly mutate each parent gene with a probability P_{mutate}. In the last step, the crossover randomly mixes two genes with a probability $P_{crossover}$. We repeat steps 2 $\sim$ 5 for N_{evo} iterations, and select the gene with the highest score as the output configuration.

To meet different users' requirements, we define our objective function using accuracy (Acc), latency (Lat) and average predictive entropy (aPE). aPE measures the quality of uncertainty estimation over a dataset of uniformly sampled random sentences of size E as: $\text{aPE} = \frac{1}{E}\sum_{e=1}^{E} - \sum_{k=1}^{K} p(y_e^k|\boldsymbol{x}_e) \log p(y_e^k|\boldsymbol{x}_e)$. $\boldsymbol{y}$ is the output probability across K classes and $\boldsymbol{x}$ is the input sentence representation. Given the randomness of the inputs, the architecture should aim for high uncertainty along with high accuracy on the test data. Therefore, the objective function can be formulated as follows:

$$Score = \alpha \times aPE + \beta \times Acc - \gamma \times Lat - PT \tag{1}$$

The overall score is a weighted average of the three different metrics parameterized by α, β and γ. The parameters enable us to define priorities in terms of algorithmic or hardware performance trade-offs while maximising the score. Additionally, there is a penalty term PT when users' constraints for $aPE^{'}$, $Acc^{'}$ and $Lat^{'}$ are not satisfied, which can be formulated as:

$$PT = \begin{cases} 0, & aPE \geq aPE^{'}, Acc \geq Acc^{'}, Lat \leq Lat^{'} \\ \gamma, & otherwise \end{cases} \tag{2}$$

In our proposed framework, users are allowed to set α, β, γ, $aPE^{'}$, $Acc^{'}$ and $Lat^{'}$ to fulfill their different needs.

4 HARDWARE ACCELERATOR

4.1 Hardware Architecture

The proposed hardware architecture for executing Bayesian transformers is presented in Figure 4. The core compute module is the adaptable processing engine (*AdaptPE*), which is followed by a shortcut (*SC*) addition, a layer normalization (*LN*) and a dropout for the post processing. We perform the regular MCD using the dropout module. The per-head MCD is implemented using the adaptable Bernoulli sampler, input and weight managers. An *SC* buffer is designed to cache the input data while performing the *SC* addition. The intermediate results are stored in the off-chip memory to reduce the resource usage of on-chip memory. We adopt block floating-point [5, 7, 23] with 16-bit mantissa and 8-bit exponent to represent both data and weights to improve hardware performance.

In *AdaptPE*, there are five different computational engines (*CE*s), namely CE_Q, CE_K, CE_V, CE_{QK} and CE_{WV}. Each *CE* is composed of a multiply-accumulate unit (*MAC*) followed by an accumulator and an *SC* adder. We design each *CE* with a demultiplexer or a multiplexer to support run-time adaptability for different data flows and computation. There are three buffers in each *AdaptPE* to cache intermediate results. A softmax module is placed between CE_{QK} and CE_{WV}. The adaptable Bernoulli sampler consists of N_{lfsr} 128-bit linear-feedback shift registers (*LFSR*s) to generate random bits. A probability (*Prob*) control module receives N_{lfsr} random bits, and outputs a Bernoulli random variable with the desired probability using extra logic.

4.2 Computational Process

The computation of the whole network is performed block by block using the *AdaptPE*s. To further reduce the on-chip memory usage,

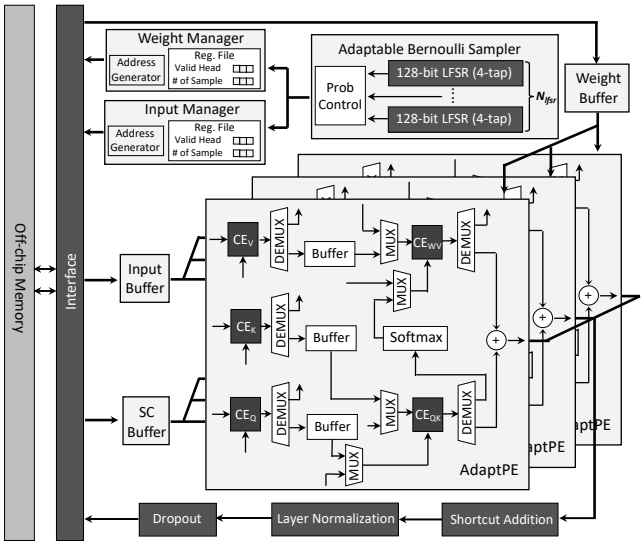


Figure 4: Hardware architecture of our design.

we divide each attention block into four chunks as shown in Figure 5(a). The first chunk contains the computation associated with the dot-product attention, including transformations and multiplication between query, key and value vectors. The second chunk includes a linear transformation *LT*, followed by the *SC* addition and the *LN*. The main computation in both third and fourth chunks are fully connected (*FC*) layers. There is an addition and an *LN* at the end of the fourth chunk. All the intermediate results between chunks are stored in the off-chip memory.

To improve hardware efficiency, our *AdaptPE* is designed to support different chunks with runtime adaptability using demultiplexers and multiplexers as shown in Figure 5. While processing the first chunk, we connect CE_Q, and CE_K with CE_{QK} to perform the multiplication between query and key vectors. The softmax module is enabled after CE_{QK}. Also, CE_V and CE_{QK} are connected with CE_{WV} to get the results of the first chunk. As both *FC* and *LT* are essentially matrix multiplications, *CE*s in *AdaptPE* can be reused for computation. Therefore, while processing the rest of chunks, CE_Q, CE_K and CE_V are used as separate engines for vector-matrix multiplication. At the same time, as both CE_{QK} and CE_{WV} adopt low parallelisms in their *MAC*, we connect them together using an extra adder as another engine. In this manner, we can achieve a balanced workload across different engines to eliminate inefficient

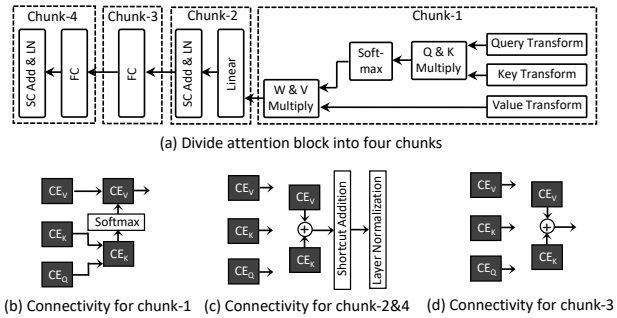


Figure 5: Adaptable computational engine.

computation. The *SC* and *LN* are also enabled for the second and forth chunks.

4.3 Skipping Sparsity

To improve the hardware performance, we design our accelerator to skip the redundant computation associated with the head, block and sample sparsity. We support redundant block and sample skipping using weight and input managers. A register in both weight and input register files is used to control the number of repeated executions in the current processing block. While running the non-Bayesian blocks, the register is set as one to ensure that the engine only runs the non-Bayesian block once. In this way, the computation of the redundant blocks can be skipped. The register is set as one until the first Bayesian block, and then is set to be the number of MC samples on Bayesian blocks. The input of the first Bayesian block will be cached, and reused for different MC samples to decrease the memory traffic.

The skipping of redundant heads is supported by using the adaptable Bernoulli sampler. To skip heads, we first perform Bernoulli sampling to generate random bits. These random bits are used as the dropout mask for MCD. By analysing the dropout mask, the positions of all the valid heads are cached in the register file. Then, the address generators in both input and weight managers calculate the addresses of valid heads using the cached positions. Therefore, only valid inputs and weights and transferred to the on-chip memory. As the Bernoulli sampling is independent to the computation, we use our Bernoulli sampler to generate random bits one block prior to the calculation to ensure that the dropout mask and the loading address are available before the processing of each block.

5 EXPERIMENTS

We implemented our EA-based framework using Python 3.7. The finetuning and evaluation of transformers were implemented using PyTorch 1.10 [16] and Huggingface's tools [30]. We used an Intel Xeon Gold 6154 CPU and a GeForce RTX 2080 Ti for CPU and GPU benchmarking. We optimized both CPU and GPU implementations to support block and sample skipping by using additional PyTorch variables to control the number of running blocks and samples. We used four pre-trained transformers: DistilBERT [21], BERT-Base [4], RoBERTa [13] and Electra [3], as baselines with respect to the proposed design space exploration and exploitation. We finetuned them for sentence classification on SST-2 and MRPC [28] datasets. The length of the input sequence was set as 128. The entropy, in terms of aPE, was measured with respect to sequences of completely random words. We set the population size as 18 and we optimized for 10 iterations. The parent, mutation and crossover sizes were all set as 6 with $P_{select} = 0.25$, $P_{mutate} = 0.5$ and $P_{crossover} = 0.5$.

5.1 Effectiveness of Framework

To visualize the effectiveness of our framework, we optimized DistilBERT on the SST-2 dataset. We iterated through and measured all the design points in terms of aPE, accuracy and latency using a GPU. The results are presented in Figure 6. We then applied our EA-based framework using three sets of optimization parameters, i.e., $\{\alpha = 1.0, \beta = 0.01, \gamma = 0.01\}$, $\{\alpha = 0.01, \beta = 1.0, \gamma = 0.01\}$ and $\{\alpha = 0.01, \beta = 0.01, \gamma = 1.0\}$, indicating different optimization

Table 1: The improvement of EA-based framework on CPU and GPU platforms.

	Evo-Mode	SST-2				MRPC			
		aPE [nats]	Acc [%]	Latency [ms]		aPE [nats]	Acc [%]	Latency [ms]	
				CPU	GPU			CPU	GPU
Bayesian DistilBERT	*Baseline*	0.312	88.64	362.17	78.33	0.304	83.62	346.93	81.80
	Evo-Opt	0.358	88.99	34.71 (10×)	6.49 (12×)	0.390	84.06	28.33 (12×)	7.35 (11×)
Bayesian BERT-Base	*Baseline*	0.241	88.07	797.32	162.66	0.076	87.01	822.03	160.59
	Evo-Opt	0.273	88.18	164.30 (5×)	31.01 (5×)	0.081	87.01	212.46 (4×)	39.13 (4×)
Bayesian RoBERTa	*Baseline*	0.132	88.50	841.07	169.00	0.062	88.97	799.51	157.68
	Evo-Opt	0.166	89.44	67.03 (13×)	14.33 (12×)	0.101	89.22	148.47 (5×)	31.56 (5×)
Bayesian Electra	*Baseline*	0.134	92.26	839.17	158.19	0.151	88.54	793.73	165.96
	Evo-Opt	0.164	92.31	73.60 (11×)	15.73 (10×)	0.429	88.72	254.77 (3×)	42.31 (4×)

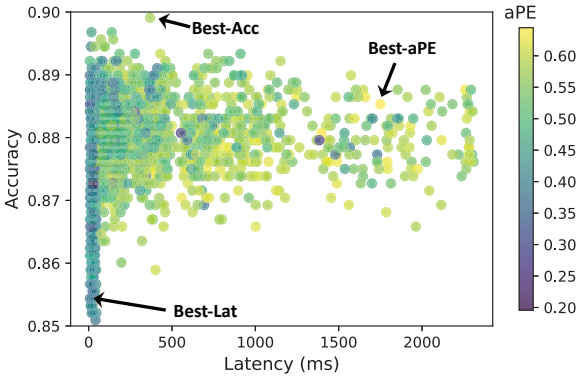
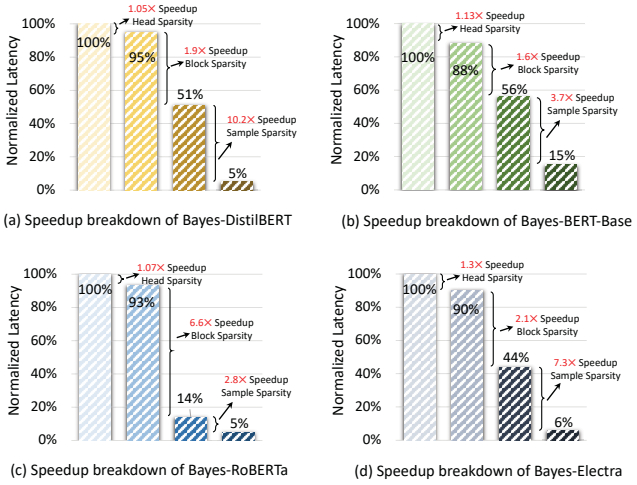


Figure 6: Design space of Bayes-DistilBERT on SST-2.



(a) Speedup breakdown of Bayes-DistilBERT

(b) Speedup breakdown of Bayes-BERT-Base

(c) Speedup breakdown of Bayes-RoBERTa

(d) Speedup breakdown of Bayes-Electra

Figure 7: Speedup breakdown of FPGA-based accelerator.

priorities for aPE, accuracy and latency respectively. As shown in Figure 6, our EA-based framework was able to find the configurations with the highest accuracy, the lowest latency and the best quality of uncertainty estimation under different priority settings.

We then applied our EA-based framework to all transformers and datasets to evaluate their performance gains on both CPU and GPU. The dropout rates were chosen from $\{0.1, 0.2, 0.4\}$ and the number of MC samples were selected from $\{2, 5, 10, 20\}$. We obtained the baseline performance of the Bayesian transformers by hand-tuning the dropout rate on the validation data and using the same dropout rate at all positions in the transformer architecture where MCD was applicable, as discussed in Section 2.1.2. The batch size was set as one to measure the latency. We set the constraints on $aPE^{'}$ and $Acc^{'}$ (Equation 2) for maintaining the baseline algorithmic performance. We set $\alpha = 0.01$, $\beta = 0.01$ and $\gamma = 1.0$ for all the models for both SST-2 and MRPC datasets. As shown in Table 1, our EA-based framework improved the latency by $4 \sim 13$ times on both CPU and GPU. Apart from the improvement in the hardware performance, we observed an increase in the algorithmic performance. For Bayesian RoBERTa on SST-2, we improved the accuracy by 0.94% and the aPE by 0.03 nats. On MRPC, Bayesian Electra and Bayesian DiltilBERT achieved 0.028 and 0.086 higher aPE compared to their baseline performance.

5.2 Performance of Hardware Architecture

We implemented our hardware architecture on a Xilinx VCU118 platform using Verilog. The design parameters P_{head}, P_{LT} and P_{QK} were set as 6, 128 and 64 respectively for a load-balanced pipeline. The resource utilization is presented in Table 2 and the final FPGA design was clocked at 182 MHz. We used the proposed framework on our FPGA-based design to optimize the configurations of Bayesian transformers. The optimization parameters of EA were kept the same as in Section 5.1. We used the SST-2 dataset to visualize the effect of different sparsity categories on our design in Figure 7. As it can be seen, the head sparsity improved the hardware performance by $1.05 \sim 1.3$ times depending on different models. Block sparsity decreased the latency by $1.6 \sim 6.6$ times. At the same time, there was $3.7 \sim 10.2$ times speedup introduced by sample sparsity. By exploiting all sparsity categories, it achieved up to 20 times speedup on both Bayes-DistilBERT and Bayes-RoBERTa.

Table 3 compares our FPGA-based design with a GPU implementation. Both implementations used the same EA optimized

Table 2: Resource utilization of the design on the FPGA.

Resources	LUTs	Registers	DSPs	BRAM
Used	822,902	1,315,368	6,144	1,391
Total	1,182,240	2,364,480	6,840	2160
Utilization	70%	55%	89%	64%

Table 3: Performance comparison of our FPGA design versus GPU implementation.

	GPU		Our Work	
Platform	GeForce RTX 2080 Ti		Xilinx VCU118	
Frequency	1.545 GHz		182 MHz	
Technology	12 nm		20 nm	
Acceleration Library	CuDNN, PyTorch 1.10		-	
Power [W]	238		45	
Model	*Bayes-DistilBert*	*Bayes-RoBERTa*	*Bayes-DistilBert*	*Bayes-RoBERTa*
Latency (Norm.) [ms]	6.49	14.33	7.88 (4.73)	14.38 (8.6)
Energy Eff. [J/Sequence]	1.54	3.41	0.35	0.64

configurations on Bayes-DistilBERT and Bayes-RoBERTa for a fair comparison. Since the input sentences are produced sequentially in real-life applications, we set the batch size to one. Table 3 shows our design is over 5 times higher energy efficiency on Bayesian DistilBERT. Although our design was slightly slower than the GPU implementation, our board adopted an older technology (20nm). If we scaled the performance by $\frac{20}{12}$ with respect to the GPU technology (12nm), our design was able to achieve lower latency as seen in the brackets in Table 3. There are three reasons for the high hardware performance of our design: *i)* an adaptable compute engine to accelerate multiple layers together using runtime adaptability, *ii)* the support of intelligent head, block and sample skipping and *iii)* the EA-based framework to exploit three categories of sparsity.

6 CONCLUSION

This paper proposes an evolutionary algorithm (EA)-based framework to exploit sparsity in Bayesian transformers. We achieve up to 13 and 12 times speedup on our optimized CPU and GPU implementations. Additionally, an adaptable hardware architecture is proposed to accelerate Bayesian transformers on an FPGA, which achieves up to 5 times higher energy efficiency than GPU implementation. In future, we aim to extend our EA-based framework to other neural networks, improve the generality of our design and explore a sampling-free method for Bayesian inference.

ACKNOWLEDGEMENT

The support of UK EPSRC grants (UK EPSRC grants EP/L016796/1, EP/N031768/1, EP/P010040/1, EP/V028251/1 and EP/S030069/1) is gratefully acknowledged.

REFERENCES

[1] H. Awano and M. Hashimoto. 2020. BYNQNet: Bayesian Neural Network with Quadratic Activations for Sampling-Free Uncertainty Estimation on FPGA. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1402–1407.
[2] Ruizhe Cai et al. 2018. VIBNN: Hardware acceleration of Bayesian neural networks. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Vol. 53. 476–488.
[3] Kevin Clark et al. 2020. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555* (2020).
[4] Jacob Devlin et al. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *CoRR* (2018).
[5] Hongxiang Fan et al. 2019. Static block floating-point quantization for convolutional neural networks on fpga. In *International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 28–35.
[6] Hongxiang Fan et al. 2021. High-Performance FPGA-based Accelerator for Bayesian Neural Networks. In *Proceedings of the 2021 ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.
[7] Jeremy Fowers et al. 2018. A configurable cloud-scale DNN processor for real-time AI. In *ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1–14.
[8] 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.
[9] Tae Jun Ham et al. 2020. Aˆ 3: Accelerating Attention Mechanisms in Neural Networks with Approximation. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 328–341.
[10] Ranganath Krishnan et al. 2020. Specifying weight priors in bayesian deep neural networks with empirical bayes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 4477–4484.
[11] Christian Leibig et al. 2017. Leveraging uncertainty information from deep neural networks for disease detection. *Scientific Reports* 7, 1 (2017), 1–14.
[12] Bingbing Li et al. 2020. FTRANS: energy-efficient acceleration of transformers using FPGA. In *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*. 175–180.
[13] Yinhan Liu et al. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).
[14] Siyuan Lu et al. 2020. Hardware Accelerator for Multi-Head Attention and Position-Wise Feed-Forward in the Transformer. *arXiv preprint arXiv:2009.08605* (2020).
[15] Rowan McAllister et al. 2017. Concrete Problems for Autonomous Vehicle Safety: Advantages of Bayesian Deep Learning. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI),*. 4745–4753.
[16] Adam Paszke et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Proceedings of the 2019 Advances in neural information processing systems (NeurIPS)* 32 (2019), 8026–8037.
[17] Aditya Prakash, Kashyap Chitta, and Andreas Geiger. 2021. Multi-Modal Fusion Transformer for End-to-End Autonomous Driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7077–7087.
[18] Alec Radford et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.
[19] Colin Raffel et al. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683* (2019).
[20] Johanna Rock, Tiago Azevedo, René de Jong, Daniel Ruiz-Muñoz, and Partha Maji. 2021. On Efficient Uncertainty Estimation for Resource-Constrained Mobile Applications. *arXiv preprint arXiv:2111.09838* (2021).
[21] Victor Sanh et al. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR* (2019).
[22] Artem Shelmanov et al. 2021. How Certain is Your Transformer?. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume (EACL)*. 1833–1840.
[23] Zhourui Song et al. 2018. Computation error analysis of block floating point arithmetic oriented convolution neural network accelerator design. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.
[24] Nitish Srivastava et al. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.
[25] Evgenii Tsymbalov et al. 2020. Dropout Strikes Back: Improved Uncertainty Estimation via Diversity Sampling. *arXiv preprint arXiv:2003.03274* (2020).
[26] Ashish Vaswani et al. 2017. Attention is all you need. In *Advances in neural information processing systems (NeurIPS)*. 5998–6008.
[27] Qiyu Wan and Xin Fu. 2020. Fast-BCNN: Massive Neuron Skipping in Bayesian Convolutional Neural Networks. In *Proceedings of the 2020 Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 229–240.
[28] Alex Wang et al. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461* (2018).
[29] Hanrui Wang et al. 2021. SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning. *IEEE International Symposium on High Performance Computer Architecture (HPCA)* (2021).
[30] Thomas Wolf et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771* (2019).
[31] Bichen Wu et al. 2020. Visual transformers: Token-based image representation and processing for computer vision. *arXiv preprint arXiv:2006.03677* (2020).
[32] Boyang Xue et al. 2021. Bayesian transformer language models for speech recognition. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 7378–7382.
[33] 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).
[34] Wangchunshu Zhou et al. 2020. Scheduled drophead: A regularization method for transformer models. *arXiv preprint arXiv:2004.13342* (2020).