ProTran: Profiling the Energy of Transformers on Embedded Platforms

Shikhar Tuli, *Student Member, IEEE,* and Niraj K. Jha, *Fellow, IEEE*

***Abstract*—Recently, automated design of efficient transformer models has attracted significant attention from both industry and academia. However, most works only focus on certain metrics while searching for the best-performing architecture. Furthermore, running traditional, complex, and large transformer models on low-compute edge platforms is a challenging task. In this work, we propose a framework to profile the hardware performance measures for a design space of transformer architectures – ProTran. Our framework can be used in conjunction with state-of-the-art neural architecture search techniques to obtain best-performing models that not only have high accuracy on the given task, but also minimize latency, energy consumption and peak power draw, that are important in most edge deployments.**

***Index Terms*—Embedded platforms, machine learning, transformer design space.**

I. Introduction

In recent years, self-attention-based transformer models [1, 2] have achieved state-of-the-art results on tasks that span the natural language processing (NLP), and recently, even the computer vision domain [3]. Increasing computational power and large-scale pre-training datasets have resulted in an explosion in the architecture size of transformer models [4], much beyond the state-of-the-art convolutional neural networks (CNNs). For instance, the Megatron Tuning-NLG [4] has 530B trainable model parameters compared to only 928M trainable model parameters in BiT (which uses ResNet-152 with every hidden layer widened by a factor of four, i.e., ResNet-152x4) [5, 6]. However, such massive transformer architectures are not amenable to be run on mobile edge devices, due to a much lower compute budget and memory size.

Even if such a large model is run on these mobile devices, it would incur an extremely high latency [7]. Smaller models may have reasonable latencies, however, they may still not meet the energy or peak power budget for running on edge devices. This could be due to a limited battery size or an intermittent power supply. Thus, there is a need for profiling and benchmarking the latency, energy and peak power consumption for a diverse set of mobile-friendly transformer architectures. This would aid frameworks to leverage hardware-aware neural architecture search (NAS) [7, 8] techniques to find the optimal architecture that maximizes model accuracy while meeting latency, energy and peak power budgets.

Several works have tried to prune transformer models to reduce the number of trainable model parameters [9, 10]. Some have also proposed novel attention mechanisms to reduce the number of trainable parameters [11, 12, 13]. Others have run NAS in a design space of transformer architectural hyperparameters to obtain efficient architectures [7, 14, 15]. However, most of these works have only shown gains in the number of model parameters or the number of floating-point operations per second (FLOPs). Such works do not consider latency, energy and power consumption in their optimization loop, while searching for the optimal transformer architecture (Wang et al. [7] only consider latency for running around 2000 transformer architectures on certain edge devices, and Li et al. [16] consider only a single FPGA). Thus, there is a need to profile not only the accuracy [8], but also the latency, energy consumption and peak power draw of transformer models on various mobile devices for inclusive design in edge-AI deployments.

Nevertheless, profiling all models in vast design spaces is a challenging endeavor. Hence, in this work, we make the following contributions:

- We implement the FlexiBERT benchmarking framework [8] with its design space of diverse transformer architectures on multiple edge-AI devices, for both training and inference. We measure the latency, energy consumption and peak power draw for the transformer models in the design space. We call this profiling framework that can obtain all hardware performance measures for a design space of transformer architectures, on a given edge platform, as ProTran.
- We leverage ProTran to train surrogate models that minimize the sample complexity of evaluation queries in the active learning loop. We leverage various regression frameworks, including Gaussian process (GP), decision tree (DT), boosted decision tree (BDT), and a state-of-the art method, i.e., BOSHNAS that exploits gradient-based optimization using backpropagation of inputs and heteroscedastic modeling [8, 17] to minimize the overall uncertainty in estimation of each measure.
- We then leverage ProTran's surrogate models for hardware performance measures along with previously proposed performance predictors [8] to run *fast* and *efficient* hardware-aware NAS that gives equivalent performing models in terms of accuracy, but with much lower latency, energy consumption and peak power draw. FlexiBERT's surrogate model is used as a preliminary accuracy predictor for our hardware-aware NAS framework for edge-AI deployments, namely, EdgeTran.

The rest of the article is organized as follows. Section II

This work was supported by NSF Grant No. —. S. Tuli and N. K. Jha are with the Department of Electrical and Computer Engineering, Princeton University, Princeton, NJ, 08544, USA (e-mail: {stuli, jha}@princeton.edu).
Manuscript received —; revised —.

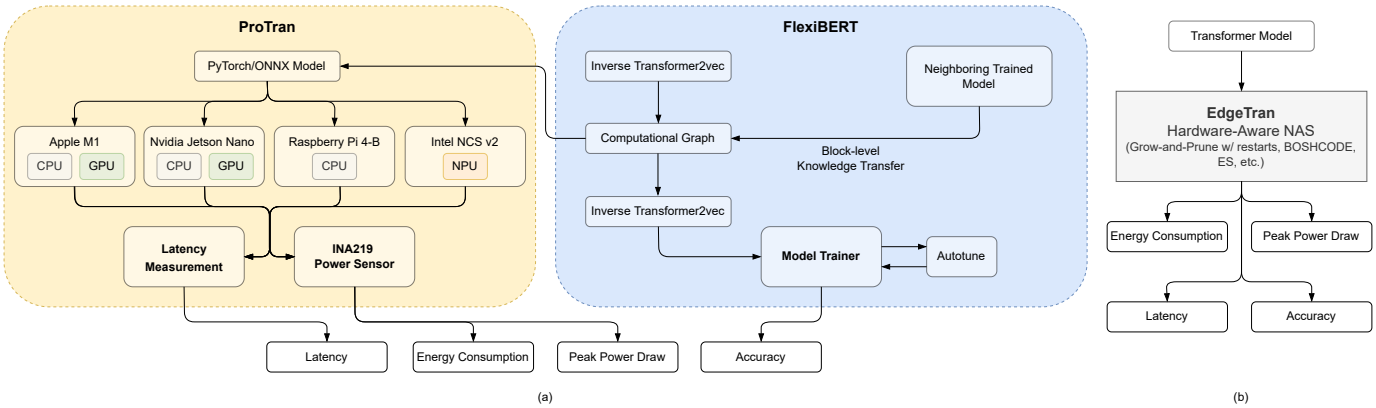

Fig. 1: Overview of the EdgeTran framework: (a) ProTran being used in conjunction with FlexiBERT for modeling accuracy along with latency, energy consumption, and peak power draw for different embedded platforms (hardware performance measures), (b) hardware-aware NAS exploiting ProTran and FlexiBERT for simultaneous optimization of accuracy with the hardware performance measures.

discusses the background and related work in hardware-aware NAS, pruning methods and profiling of transformer models. Section III motivates the need for the ProTran framework using a toy example in our design space. Section IV presents the ProTran framework in detail. Section V describes the experimental setup and the baselines considered. Section VI discusses the results. Finally, Section VII concludes this article.

II. Background and Related Work

This section introduces the relevant background and related works in the fields of hardware-aware NAS, pruning methods and profiling of transformer architectures.

A. Transformer Architectures

Various transformer architectures have been proposed in the past. BERT is one of the most popular transformer architectures that is widely used for language modeling [2]. Its variants leverage mechanisms other than the vanilla self-attention [18] to either optimize performance or reduce model size and complexity. These include – RoBERTa [19] that implements robust pre-training techniques, ConvBERT [20] that uses one-dimensional convolutional operations, MobileBERT [21] that uses bottleneck structures and multiple feed-forward stacks, SqueezeBERT [13] that uses grouped convolution operations to approximate the feed-forward stack, etc. Further, architectures like FNet [11] and LinFormer [12] use the Fourier transform or a low-rank approximation, respectively, of the self-attention operation to aid efficiency and reduce the number of model parameters.

Many works have tried to devise design spaces in order to search for optimal architectural design decisions in a unified manner. For instance, SchuBERT proposes a design space of transformer architectures [22]. However, this work does not consider different types of attention operations and only has *homogeneous* models in its design space. In this work, we leverage FlexiBERT, a state-of-the-art benchmarking framework for diverse transformer architectures, which incorporates most popularly used attention operations in a design space of *heterogeneous* and *flexible* transformer architectures [8]. We model the latency, energy and peak power for transformer architectures on a diverse set of embedded platforms, thanks to the FlexiBERT framework.

B. Hardware-Aware NAS

NAS basically searches for the architecture which attains the best accuracy on a specified dataset. However, NAS alone is hardly of much use if the best-performing transformer cannot be run on the hardware at hand (or does not meet the hardware performance constraints). Hence, recent works have focused on hardware-aware NAS which directs the architecture search for a target platform. ChamNet proposed accuracy and resource (latency and energy) predictors, and leveraged GP-based Bayesian optimization (GP-BO) to find the optimal CNN architecture for a given platform [23]. Some works have also proposed *simultaneous* co-design of the hardware and the software design decisions [24, 25, 26]. However, these works are only limited to CNN design spaces.

HAT [7], a recent framework for hardware-aware NAS, trains a large transformer model first and then uses latency feedback to obtain a sub-model for the given hardware platform. However, all sub-models are *homogeneous* and have constant dimensionality in each encoder layer. Further, this work uses a static training recipe, which may not be optimal for every sub-model. Lastly, the design space is highly restricted, which has been shown to lead to marginal gains [26]. Instead, other NAS techniques can be leveraged for superior and efficient search of the optimal model in a diverse set of transformer architectures [8, 23, 27]. Fig. 1 presents EdgeTran, that exploits ProTran, and leverages the FlexiBERT framework (along with a hardware-aware NAS pipeline) to implement block-level grow-and-prune with random warm restarts of transformer architectures to obtain the best-performing model in terms of accuracy, latency, energy and peak power consumption. However, we leverage FlexiBERT's accuracy predictor as a preliminary surrogate model for EdgeTran's NAS.

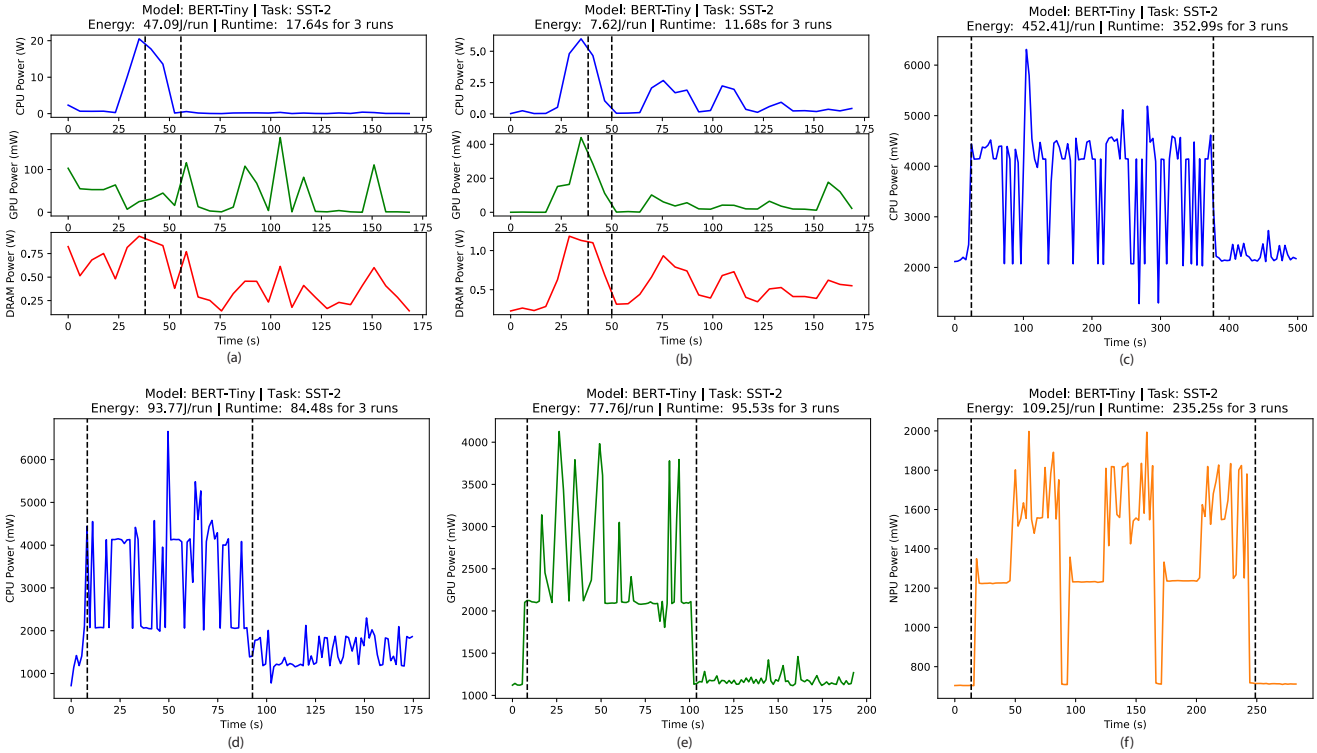

Fig. 2: Power consumption from different sources (CPU, GPU or DRAM) for different platforms: (a) Apple M1 SoC on 8-core CPU, (b) Apple M1 SoC on 8-core GPU, (c) Raspberry Pi quad-core CPU, (d) Nvidia Jetson Nano (with 2GB unified memory) quad-core CPU, (e) Nvidia Jetson Nano GPU, and (f) Intel Neural Compute Stick NPU. One run corresponds to a full pass of running inference of the BERT-Tiny [28] model on the SST-2 [29] task for the entire dataset.

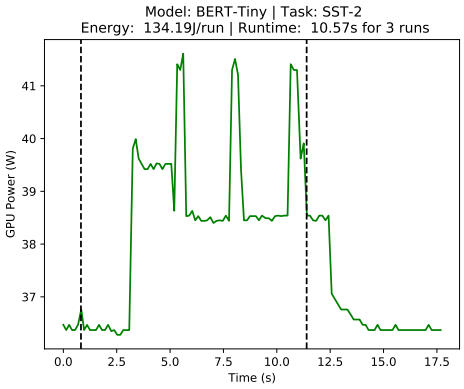

Fig. 3: Power consumption for Nvidia A100 GPU. One run corresponds to a full pass of running inference of the BERT-Tiny [28] model on the SST-2 [29] task for the entire dataset.

C. Energy Profiling of Transformer Models

Profiling the energy consumption of any ML model is a challenging task. This is because extracting the energy consumed only by the processes of training or running inference for an ML model is non-trivial. Further, if the design space is large, running training or inference for each model could take drastically long times. Nevertheless, many previous works have profiled the energy consumption of ML architectures. ChamNet trains predictors for the energy consumption for various CNNs in its design space, on different hardware platforms under various latency constraints [23]. FTRANS shows the energy and power consumption for different transformer architectures on an FPGA [16]. However, no NAS approach on transformer architectures has accounted for energy and peak power consumption in the past [7, 8]. Thus, there is a need for light-weight surrogate models for energy and peak power estimation of a diverse set of transformer architectures on various edge-AI platforms. This would not only aid energy-aware NAS for transformer models, but also efficient co-design for optimal edge deployments.

III. Motivation

This section motivates the need for rigorous profiling of transformer architectures on diverse edge-AI platforms and the development of light-weight surrogate models for various hardware performance measures.

A. Energy Reduction on Mobile Platforms

Fig. 2 plots the power consumption for running model inference for BERT-Tiny [28] on the SST-2 task [29], considering different hardware platforms. Figs. 2(a) and (b) show the central processing unit (CPU), graphics processing unit (GPU), and dynamic random access memory (DRAM) power consumption on an iPad (with the Apple M1 SoC) [30] Fig. 2(c) shows the power consumption for the Raspberry Pi CPU. Fig. 2(d) and (e) show the power consumption for the

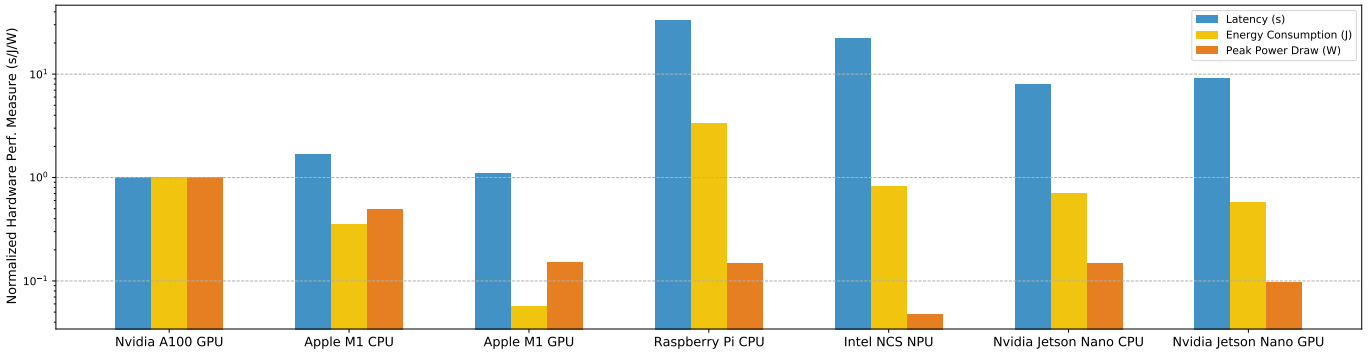

Fig. 4: Gains in different hardware performance measures for diverse embedded platforms. Results have been normalized against those for the baseline, i.e., the Nvidia A100 GPU.

Nvidia Jetson Nano, when run on the CPU and the GPU, respectively. Finally, Fig. 2 shows the power consumption for the Intel Neural Compute Stick (NCS) v2. These power-draw profiles can be compared against that of a regular GPU in Fig. 3(c) (we use Nvidia A100 for the baseline GPU). These figures show that the mobile platforms has much lower power consumption throughout their operation when compared to the A100 GPU. These profiles also highlight diverse power draw characteristics and peak power consumption for different platforms.

As can be seen from Fig. 4, the Apple M1 SoC on its integrated GPU outperforms the traditional A100 GPU in terms of energy and peak power consumption, and is also close in terms of latency. The total energy consumption being the minimum among all platforms in consideration. We see 17.61× reduction in energy consumption and 6.56× reduction in peak power draw, while having only 10.58% higher latency per run, for the Apple M1 SoC running on its integrated GPU when compared to the Nvidia A100 GPU. On the other hand, the Intel Neural Compute Stick has the minimum peak power draw, i.e., 21× reduction in peak power draw, with, however, 22.3× higher latency compared to the Nvidia A100 GPU. This shows a diverse set of latency, energy and peak power draw profiles for different platforms that might fit various requirements or constraints in hardware performance for assorted edge-AI deployments. This motivates the need for profiling the latency, energy, and peak power consumption of various transformer architectures on a diverse set of embedded platforms. This would aid efficient design of transformer architectural design decisions for different edge-AI deployments.

B. Challenges and Proposed Solutions

The example shown in Fig. 2 only runs inference of a toy transformer architecture, namely BERT-Tiny. For larger architectures, and even for training, much more memory is often required. Such large models (e.g., BERT-Large) may not fit into the memory of the integrated CPU or GPU. To counter this, gradient accumulation over multiple subsets of the mini-batch could be exploited, however, at the cost of increased latency. A challenge that remains is to efficiently search for optimal transformer architectures that can fit into the memory while also not compromising on accuracy, or the need for gradient accumulation that hurts latency. For this, we can experiment different approaches, including NAS, grow-and-prune, etc. Further, we can explore quantization and reduction of the activations inside the model to alleviate memory bottlenecks [7].

TABLE I: Design space description. Super-script (j) depicts the value for layer j.

Design Element	**Allowed Values**
Number of encoder layers (l)	$\{2, 4, 6, 8, 10, 12\}$
Type of attention operation used (o^j)	$\{sa, l, c\}$
Number of operation heads (n^j)	$\{2, 4, 8, 12\}$
Hidden size (h^j)	$\{128, 256\}$
Feed-forward dimension (f^j)	$\{256, 512, 1024, 2048, 3072, 4096\}$
Number of feed-forward stacks	$\{1, 2, 3\}$
Operation parameters (p^j):	
if $o^j = sa$	Self-attention type: $\{sdp\ , wma\}$
else if $o^j = l$	Linear transform type: $\{dft\ , dct\}$
else if $o^j = c$	Convolution kernel size: $\{5, 9\}$

IV. The ProTran Framework

To counteract the above challenges, we present the ProTran framework in this section.

A. Transformer Design Space

As proposed by Tuli et al. [8], the design space of heterogeneous transformer architectures is immense. There are a diverse set of operations that can be added to a transformer architecture. We leverage the modular architectures in Flexi-BERT for our design space. The traditional BERT model is composed of multiple layers, each containing a bidirectional multi-headed self-attention (denoted by sa) module followed by a feed-forward module. Several modifications have been proposed to the original encoder, primarily to the attention module. This gives rise to a richer design space. We consider weighted multiplicative attention (wma)-based self-attention in addition to scaled dot-product (sdp)-based operations [31].

We also incorporate linear transform (l)-based attention in FNet and dynamic-span-based convolution (c) in ConvBERT, in place of the vanilla self-attention mechanism. Whereas the original FNet implementation uses discrete Fourier transform

(dft), we also consider discrete cosine transform (dct). Our design space allows variable kernel sizes for convolution-based attention. Consolidating different attention module types that vary in their computational costs into a single design space enables the models to have inter-layer variance in expression capacity. Inspired by MobileBERT [21], we also consider architectures with multiple feed-forward stacks. We summarize the entire design space with the range of each operation type in Table I.

However, unlike FlexiBERT's design space, we use a wider range of values for each of the hyperparameters, especially for even more diverse architectures. This results in architectures that are substantially different from traditional BERT-like ones. Further, we make the architectures in the FlexiBERT pipeline even more *heterogeneous*, i.e., instead of all attention operations in an encoder layer to be the same, we now allow an encoder layer to have different types of operations (we call it the FlexiBERT v2.0 design space). In other words, where FlexiBERT only allows, say, sa attention heads in an encoder layer, our design space allows other attention types as well (from wma, c, or l). Further, weight transfer has also been updated at an 'attention-head level', i.e., entire encoder layer hyperparameters are not required to be the same; if some of the attention heads are alike in two neighboring models, the weights for those attention heads can be transfered. Considering all the possible hyperparameter values given in Table I, we generate a design space with 1.69×10^{88} architectures, much larger than any previous work.

B. Transformer Embeddings

The FlexiBERT pipeline leverages the `Transformer2vec` embedding scheme to form dense embeddings for the transformer architectures in the design space. However, training these embeddings is computationally expensive. We thus use a *naive* and *sparse* embedding scheme that we present next.

For the selected ranges of hyperparamters in our design space (see Table I), we generate embeddings that are 37 dimensional as follows:

- The first entry in the embedding is the number of encoder layers in the current transformer model. In other words, for the embedding for a transformer architecture, e, e_1 is the number of encoder layers in the architecture.
- For an encoder layer, l, $e_{1+3(l-1)}$ is the hidden dimension for that encoder layer. The hidden dimension of each attention head is equally divided based on the hidden dimension of the encoder layer and the number of attention heads in the encoder layer.
- For an encoder layer, l, $e_{2+3(l-1)}$ is the index of the feed-forward operation formed by the range of feed-forward layers possible for the given design space. For the six possible hidden dimensions in the feed-forward layers, there can be a stack of up to three layers, thus giving rise to 258 feed-forward operation types for every encoder layer.
- For an encoder layer, l, $e_{3+3(l-1)}$ is the index of the attention head operation in the list of multi-head operations types. These are determined by the number of attention heads selected for that encoder layer, the type of each attention head and their respective combinations with replacement, giving rise to 21805 attention head operation types possible for every encoder layer.
- For models that are less than 12 layers deep, the respective entry in their embeddings are set to zero.

We leave incorporation of `Transformer2vec` embeddings into our design space for the future.

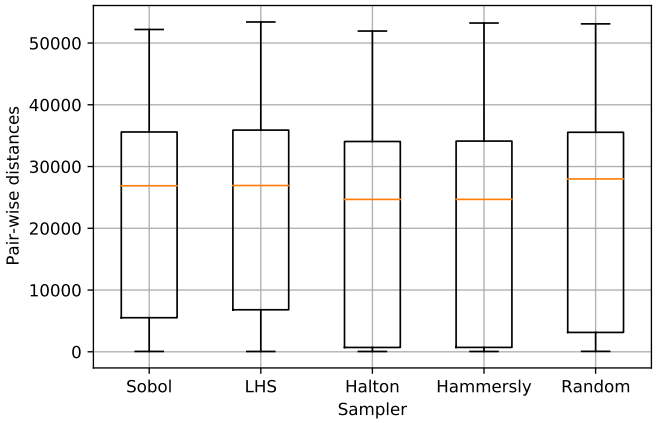

Fig. 5: Box plot for pairwise distances of 256 sampled embeddings from different sampling schemes.

C. Initial Sampling

Any regressor that is used for an active learning pipeline requires an initial seed dataset for prediction of further queries that need to be explored. Intuitively, this dataset should be as representative of the design space as possible. For this, we test various low-discrepancy sequence sampling strategies [32]. Fig. 5 shows a boxplot of pairwise distances between embeddings from sampling methods, namely, Sobol sampling, Latin hypercube sampling (LHS), Halton sampling, Hammersly sampling, and the typical Random sampling. We use LHS method for our experiments since it results in the highest first quartile of pairwise distances between the embeddings of the sampled architectures. This would mean that the probability of having divergent points in the sampled set would be maximized under this sampling scheme. Other measures could be explored in order to find the sampling scheme that results in the maximal diversity of the initial sampled architectures [33].

D. Learning the Performance Predictor

Once the initial samples are evaluated for all the hardware measures (see SectionV-A), we need to learn a performance predictor that takes in the transformer embedding as an input and predicts the hardware performance measures. This predictor, also a surrogate for the performance estimation, can then eventually be leveraged to query novel models in the design space in order to further improve the performance estimation. This strategy is employed in an active learning scenario so that the number of queried models for evaluation is minimized.

For the active learning loop, we experimented with several regression schemes – GP, DT, BDT, and a state-of-the art

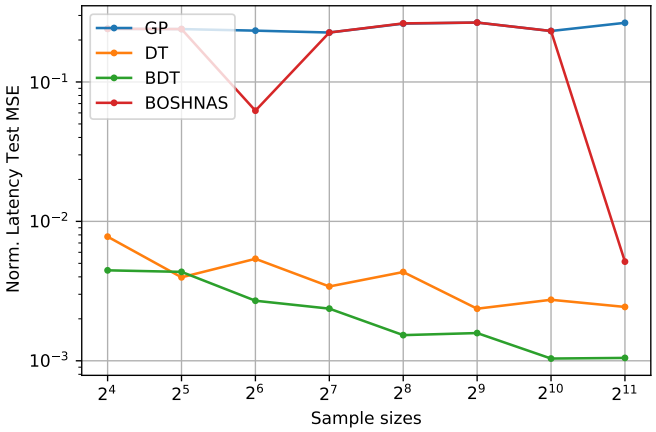

Fig. 6: Test MSE on the normalized latency values for different sample sizes while using various regressors.

TABLE II: Selected batch size and number of samples for convergence for different platforms.

Platform	Batch size	Number of samples
Nvidia A100 GPU	128	55
Apple M1 CPU	32	78
Apple M1 GPU	32	138
Raspberry Pi CPU	1	
Intel NCS NPU	1	
Nvidia Jetson Nano CPU	1	164
Nvidia Jetson Nano GPU	1	22

method, i.e., BOSHNAS that exploits gradient-based optimization using backpropagation of inputs and heteroscedastic modeling [8, 17] to minimize the overall uncertainty in estimation of the prediction measures. GP and BOSHNAS directly indicate the epistemic uncertainty in predictions, and thus, the next query in the active learning loop can be selected as the model with the highest uncertainty. The uncertainty in estimation for BDT was computed as the standard deviation in the predictions of each decision tree for every output hardware performance measure. DT, however, can not model the uncertainty in performance prediction.

We test these regressors on a dataset of inference latency on the Nvidia A100 GPU (see Section V-A), where we generate a randomly sampled set of models and find the latency of inference for the SST-2 task in the GLUE benchmarking suite [29]. We then take smaller subsets of this dataset and then divide each subset into train/test (80-20) splits to check the mean-squared error (MSE) in the prediciton on the test set after training the regressor on the test set. As presented in Fig. 6, BDT has the lowest prediction error on the test set for different sample sizes. Thus, we use BDT for our active learning loop, while training the performance predictors for all platforms. Table II presents the sample sizes required to converge the BDT for different platforms. Convergence was reached when the test MSE was below 0.3% for each of latency, energy, and peak power draw in the normalized form (e.g. latency values are divided by the maximum latency in the dataset to obtain all values between zero and one).

Fig. 7 plots the predicted latency with the real latency for the

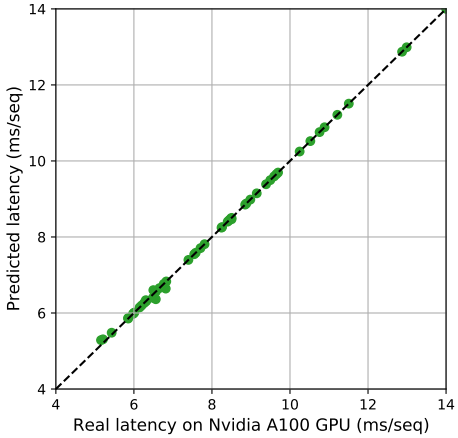

Fig. 7: Predicted and real latencies on the template Nvidia A100 GPU dataset. The size of the sample set is 256 architectures. Latencies are reported per sequence in the SST-2 task.

BDT regressor on the template Nvidia A100 dataset of latency measurements. The plot shows that the predicted latency is very close to the real latency.

V. Experimental Setup

A. Hardware Platforms

B. Baselines

VI. Results

A. Performance Profiles

B. Hardware-Aware NAS using EdgeTran

VII. Conclusion

Acknowledgments

The simulations presented in this article, along with experiments on the A100 GPUs, were performed on computational resources managed and supported by Princeton Research Computing at Princeton University. Various hardware platforms were sponsored by the Department of Electrical and Computer Engineering at Princeton University.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Int. Conf. Neural Information Processing Systems*, vol. 30, 2017, pp. 5998–6008.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, vol. 1, 2019, pp. 4171–4186.

[3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learning Representations*, 2021.

[4] S. Smith, M. Patwary, B. Norick, P. LeGresley, S. Rajbhandari, J. Casper, Z. Liu, S. Prabhumoye, G. Zerveas, V. Korthikanti, E. Zheng, R. Child, R. Y. Aminabadi, J. Bernauer, X. Song, M. Shoeybi, Y. He, M. Houston, S. Tiwary, and B. Catanzaro, "Using DeepSpeed and Megatron to train Megatron-Turing NLG 530b, A large-scale generative language model," *CoRR*, vol. abs/2201.11990, 2022.

[5] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, "Big Transfer (BiT): General visual representation

learning," in *Proc. European Conference on Computer Vision*, 2020, pp. 491–507.
[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2016, pp. 770–778.
[7] H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han, "HAT: Hardware-aware transformers for efficient natural language processing," in *Proc. 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 7675–7688.
[8] Anonymous, "FlexiBERT: Expanding the flexibility of transformer architectures and exploring a heterogeneous design space," in *Proc. Int. Conf. on Machine Learning*, 2022, in review.
[9] M. A. Gordon, K. Duh, and N. Andrews, "Compressing BERT: studying the effects of weight pruning on transfer learning," *CoRR*, vol. abs/2002.08307, 2020.
[10] Z. Yan, H. Wang, D. Guo, and S. Han, "MicroNet for efficient language modeling," in *Proc. Int. Conf. Neural Information Processing Systems 2019 Competition and Demonstration Track*, vol. 123, 08–14 Dec 2020, pp. 215–231.
[11] J. Lee-Thorp, J. Ainslie, I. Eckstein, and S. Ontañón, "FNet: Mixing tokens with Fourier transforms," *CoRR*, vol. abs/2105.03824, 2021.
[12] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," *CoRR*, vol. abs/2006.04768, 2020.
[13] F. Iandola, A. Shaw, R. Krishna, and K. Keutzer, "SqueezeBERT: What can computer vision teach NLP about efficient neural networks?" in *Proc. SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*. Online: Association for Computational Linguistics, Nov. 2020, pp. 124–135.
[14] J. Xu, X. Tan, R. Luo, K. Song, J. Li, T. Qin, and T.-Y. Liu, "NAS-BERT: Task-agnostic and adaptive-size BERT compression with neural architecture search," in *Proc. 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, p. 1933–1943.
[15] Y. Yin, C. Chen, L. Shang, X. Jiang, X. Chen, and Q. Liu, "AutoTinyBERT: Automatic hyper-parameter optimization for efficient pre-trained language models," in *Proc. 59th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, Aug. 2021, pp. 5146–5157.
[16] B. Li, S. Pandey, H. Fang, Y. Lyv, J. Li, J. Chen, M. Xie, L. Wan, H. Liu, and C. Ding, "FTRANS: Energy-efficient acceleration of transformers using FPGA," in *Proc. ACM/IEEE International Symposium on Low Power Electronics and Design*, 2020, p. 175–180.
[17] S. Tuli, S. R. Poojara, S. N. Srirama, G. Casale, and N. R. Jennings, "COSCO: Container orchestration using co-simulation and gradient based optimization for fog computing environments," *IEEE Trans. Parallel and Distributed Systems*, vol. 33, no. 1, pp. 101–116, 2021.
[18] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," in *Proc. Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, vol. 2, 2018, pp. 464–468.
[19] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019.
[20] Z.-H. Jiang, W. Yu, D. Zhou, Y. Chen, J. Feng, and S. Yan, "ConvBERT: Improving BERT with span-based dynamic convolution," in *Proc. Int. Conf. Neural Information Processing Systems*, vol. 33, 2020, pp. 12 837–12 848.
[21] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "MobileBERT: A compact task-agnostic BERT for resource-limited devices," in *Proc. 58th Annual Meeting of the Association for Computational Linguistics*, Jul. 2020, pp. 2158–2170.
[22] A. Khetan and Z. Karnin, "schuBERT: Optimizing elements of BERT," in *Proc. 58th Annual Meeting of the Association for Computational Linguistics*, Jul. 2020, pp. 2807–2818.
[23] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia, P. Vajda, M. Uyttendaele, and N. K. Jha, "Chamnet: Towards efficient network design through platform-aware model adaptation," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition*, June 2019.
[24] Y. Lin, M. Yang, and S. Han, "NAAS: Neural accelerator architecture search," *CoRR*, vol. abs/2105.13258, 2021.
[25] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "MCUNet: Tiny deep learning on IoT devices," in *Proc. Int. Conf. Neural Information Processing Systems*, vol. 33, 2020, pp. 11 711–11 722.
[26] S. Tuli, C. H. Li, R. Sharma, and N. K. Jha, "CODEBench: A neural architecture and hardware accelerator co-design framework," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 2022, in review.
[27] X. Dai, H. Yin, and N. K. Jha, "NeST: A neural network synthesis tool based on a grow-and-prune paradigm," *IEEE Trans. Computers*, vol. 68, no. 10, pp. 1487–1497, 2019.
[28] I. Turc, M. Chang, K. Lee, and K. Toutanova, "Well-read students learn better: The impact of student initialization on knowledge distillation," *CoRR*, vol. abs/1908.08962, 2019.
[29] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *Proc. EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, Nov. 2018, pp. 353–355.
[30] Apple. (2020) Apple unleashes M1. [Online]. Available: https://www.apple.com/newsroom/2020/11/apple-unleashes-m1/
[31] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Sep. 2015, pp. 1412–1421. [Online]. Available: https://aclanthology.org/D15-1166
[32] H. Niederreiter, *Random number generation and quasi-Monte Carlo methods*. SIAM, 1992.
[33] E. K. Tang, P. N. Suganthan, and X. Yao, "An analysis of diversity measures," *Machine Learning*, vol. 65, no. 1, pp. 247–271, Oct 2006. [Online]. Available: https://doi.org/10.1007/s10994-006-9449-2

Shikhar Tuli received the B. Tech. degree in electrical and electronics engineering from the Indian Institute of Technology (I.I.T.) Delhi, India, with a department specialization in very large scale integration (VLSI) and embedded systems. He is currently pursuing a Ph.D. degree at Princeton University in the department of electrical and computer engineering. His research interests include deep learning, edge artificial intelligence (AI), hardware-software co-design, brain-inspired computing and smart healthcare.

Niraj K. Jha (Fellow, IEEE) received the B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology (I.I.T.) Kharagpur, Kharagpur, India, in 1981, and the Ph.D. degree in electrical engineering from the University of Illinois at Urbana–Champaign, Champaign, IL, USA, in 1985. He is a Professor in the Department of Electrical and Computer Engineering, Princeton University. He has coauthored five widely used books. He has published more than 460 papers (h-index: 81). He has received the Princeton Graduate Mentoring Award. His research has won 14 best paper awards, six award nominations, and 23 patents. He was given the Distinguished Alumnus Award by I.I.T., Kharagpur. He has served as the Editor-in-Chief of TVLSI and an associate editor of several journals. He has given several keynote speeches in the areas of nanoelectronic design/test, smart healthcare, and cybersecurity. He is a fellow of ACM. His research interests include smart healthcare, cybersecurity, and machine learning algorithms/architectures.