IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 9 June 2024, accepted 21 June 2024, date of publication 5 September 2024, date of current version 13 September 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3420232

RESEARCH ARTICLE

Skipformer: Evolving Beyond Blocks for Extensively Searching On-Device Language Models With Learnable Attention Window

MATTHEW BODENHAM[1], **(Graduate Student Member, IEEE), AND JAEHA KUNG**[2], **(Member, IEEE)**

[1]Department of Electrical Engineering and Computer Science, DGIST, Daegu 42988, Republic of Korea
[2]School of Electrical Engineering, Korea University, Seoul 02841, Republic of Korea

Corresponding author: Jaeha Kung (jhkung@korea.ac.kr)

This work was supported in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP) funded by [Ministry of Science and ICT (MSIT)] under Grant RS-2023-00229849, and in part by the National Research Foundation of Korea (NRF) funded by MSIT under Grant NRF-2023R1A2C2006290.

ABSTRACT Deployment of language models to resource-constrained edge devices is an uphill battle against their ever-increasing size. The task transferability of language models makes deployment to the edge an attractive application. Prior neural architecture search (NAS) works have produced hardware-efficient transformers, but often overlook some architectural features in favor of efficient NAS. We propose a novel evolutionary NAS with large and flexible search space to encourage the exploration of previously unexplored transformer architectures. Our search space allows architectures to vary through their depth and skip connections to transfer information anywhere inside the architecture; Skipformer, the top searched model, displays these novel architectural features. To further increase Skipformer efficiency, we learn a CUDA-accelerated attention window size at each self-attention layer during training. Skipformer achieves 23.3% speed up and requires 19.2% less memory on NVIDIA Jetson Nano with negligible accuracy loss on GLEU benchmark compared to GPT-2 Small.

INDEX TERMS Language models, neural architecture search, on-device inference, transformers.

I. INTRODUCTION

Transformer [1] derived language models have made great strides in natural language processing (NLP), with state-of-the-art methods demonstrating strong performance across multiple tasks [2], [3], [4]. However, deployment of state-of-the-art language models to the edge has become unfeasible due to the ever-increasing memory and compute demands that are necessary for inference. Still, deployment of models to the edge has economical and end-user benefits, i.e., data is no longer required to be sent to remote servers for processing, reducing latency and protecting data privacy. Therefore, improvements in the computational and memory efficiency of language models must be pursued for real-time inference and reduced storage space on edge devices.

In parallel with the development of language models, neural architecture search (NAS) has shown great success in deep learning research in discovering more efficient architectures while maintaining accuracy [5], [6], [7]. Hardware-aware NAS (HW-NAS) improves NAS by targeting specific hardware devices for more optimized deployment to edge devices [8], [9], [10]. While most previous NAS works

The associate editor coordinating the review of this manuscript and approving it for publication was Yizhang Jiang.

target CNN tasks, NAS for transformer architectures has been increasing in interest [11], [12]. NAS methods targeting various hardware devices in translation [12] and language modeling [13] have also been presented. However, these methods often feature constricted search spaces in favor of reducing search time. Restricted search spaces leave many potentially promising architectures unexplored. For instance, block-based search spaces are an efficient method of reducing search time, but they assume that repeatable uniform architectures are most optimal.

In this work, we propose searching for language models using a large unrestricted *chain-based* search space. Our search space allows for a continuously changing architecture throughout the entire depth of the model, in combination of *unrestricted skip connections* that can transverse many layers. We propose a multi-stage evolutionary algorithm, which first aggressively generates new parent models. Then, in the next stage, only top-performing parent models are selected and fine-tuned with a couple of mutations allowed. The objective is to find a small language model that runs faster without accuracy loss on a memory- and power-bounded edge device, i.e., NVIDIA Jetson Nano, compared to the baseline model, i.e., GPT-2 Small [3]. To further improve the model's efficiency, we introduce a learnable attention window with a custom CUDA kernel to reduce computation time associated with the self-attention '$QK^\intercal$'.

Our contributions can be summarized as the following:

- **Edge-aware NAS for language modeling** – We present a multi-stage and latency-aware evolutionary search that encourages the search to consider many architectural parameters and layer configurations. Each stage in the proposed search encourages model diversity to find potential unexplored strong candidates.
- **Learnable self-attention window size** – During the search, attention window sizes are learned by gradient descent using our *tanh*-based masking function, improving both model accuracy and compute latency with smaller window sizes over the prior work. A custom CUDA kernel is implemented for real-world speed-up.
- **Fine-tuning searched models to other NLP tasks** – The generalizability of the searched model is verified by fine-tuning it to other natural language understanding (NLU) tasks.

II. BACKGROUND

A. NATURAL LANGUAGE PROCESSING

Natural language processing (NLP) is a multidisciplinary field that focuses on enabling computers to understand, interpret, and generate natural language [14]. NLP encompasses a wide range of tasks, including text classification, named entity recognition, sentiment analysis, machine translation, and question answering [15]. NLP techniques have numerous practical applications across various domains, including healthcare, finance, customer service, and education. Over the years, NLP has witnessed significant advancements driven by the availability of large-scale datasets, computational resources, and breakthroughs in machine learning algorithms [16]. Deep learning techniques, such as recurrent neural networks (RNNs), convolutional neural networks (CNNs), and transformer-based models, have played a pivotal role in advancing the state-of-the-art in NLP [17]. Recently, pre-trained language models, such as BERT [2] and GPT [3], [18], have demonstrated remarkable performance across a wide range of NLP tasks, leading to the emergence of transfer learning as a dominant paradigm in NLP [19], [20]. Despite these advancements, NLP still faces challenges, including handling ambiguity, understanding context, and addressing biases present in training data [21].

B. LANGUAGE MODELING

Language modeling is a task in NLP which aims to capture the statistical properties and structural dependencies inherent in natural language texts [14]. It plays a crucial role in various NLP applications, including machine translation, speech recognition, sentiment analysis, and text generation [22]. The primary objective of language modeling is to estimate the probability distribution of sequences of words or characters (called tokens) in a given language. This estimation allows language models to generate coherent and contextually relevant text, enabling them to understand and produce human-like language. Considering the incomplete sentence: **"I enjoy taking long walks in the"**. A language model trained on a dataset might predict the next word to be "**park**", "**woods**", or "**countryside**". Patterns learned from the training data allow the model to predict the next token using probability.

Recently, transformer-based architectures [23] have demonstrated superior performance in various NLP tasks, including language modeling. Pre-trained large language models (LLMs), which are trained on large-scale text corpora using unsupervised learning objectives, have emerged as a dominant paradigm in language modeling [19]. Pre-trained models such as BERT [20] and GPT [18] have demonstrated remarkable capabilities in capturing contextual information and understanding semantic nuances in text. These models serve as the foundation for transfer learning in NLP, allowing fine-tuning on specific downstream tasks with minimal labeled data. Downstream tasks include text classification, question answering, and language understanding.

C. TRANSFORMER ARCHITECTURE

Self-attention mechanism is the core of the transformer architecture, which allows each token in the input sequence to attend to all other tokens simultaneously. This enables the model to weigh the importance of each token's contextual information dynamically, without being constrained by sequential processing. Self-attention is computed by generating query, key, and value vectors for each token and then calculating attention scores between token pairs. This

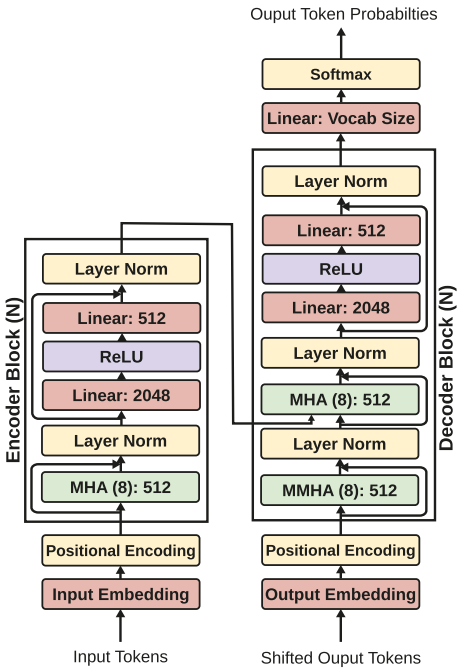

FIGURE 1. Original transformer architecture for translation tasks [1].

process is done in the multi-head attention layer shown in Figure 1.

The complexity of transformers can be analyzed using Big O notation by breaking a transformer into its key operations.

- **Self-attention mechanism:** This operation computes attention scores between all pairs of tokens in the input sequence. For each token, it computes a weighted sum of the values of all other tokens based on their relevance scores. The time complexity of this operation is $O(N^2 \cdot D)$, where N is the sequence length and D is the dimensionality of the feature vectors.
- **Position-wise feedforward networks (FFN):** After the self-attention mechanism, each token's representation passes through a position-wise feedforward network (FFN), which consists of fully connected layers with ReLU/GeLU activations. The time complexity of this operation is $O(N \cdot D_{\text{ff}})$, where D_{ff} is the dimensionality of the feedforward layer.

In the case of transformers, the dominant term usually arises from the self-attention mechanism, resulting in a simplified time complexity of $O(N^2 \cdot D)$. However, if D is small or remains constant, the complexity can be further simplified to $O(N^2)$. This simplification is often used in practice to analyze the scalability of transformer models with respect to input sequence length.

D. NEURAL ARCHITECTURE SEARCH

Neural architecture search (NAS) is a dynamic and rapidly evolving field within machine learning that focuses on automating the design of neural network architectures. Traditional approaches to developing neural network architectures often rely heavily on manual design, which can be time-consuming and requires significant domain expertise. NAS aims to mitigate these challenges by using optimization algorithms to discover optimal architectures automatically, tailored for specific tasks. NAS can be broken down into three components [24], [25]:

1) **Search space**: This defines the range of possible architectures that the search algorithm can explore. It includes choices about the number of layers, types of layers, layer sizes, connection types, and other architectural features.
2) **Search strategy**: This refers to the algorithm used to explore the search space. Common strategies include random search, reinforcement learning, evolutionary algorithms, and gradient-based methods. Each has its strengths and trade-offs in terms of speed, computational cost, and ability to find optimal solutions.
3) **Performance estimation**: This involves evaluating the performance of architectures generated during the search. Because training each model fully is computationally expensive, techniques such as learning curve extrapolation, network morphing, and one-shot models are often used to estimate performance quickly.

Deploying sophisticated models such as language models to edge devices presents unique challenges. Edge devices, which include smartphones, IoT devices, and other consumer electronics, typically have strict constraints on power, memory, and processing capabilities. To optimize language models for these platforms, hardware-aware neural architecture search (HW-NAS) has emerged as a key technology.

III. RELATED WORK

Direct deployment of LLMs to edge/mobile devices is limited by resource constraints, i.e., compute capability and memory capacity [26]. Energy efficiency is another critical factor for edge devices, as they are often battery powered, so power consumption must be reduced to the bare minimum to preserve battery life. Thus, when designing language models for edge, the following features must be considered: number of parameters, inference speed, energy consumption, and hardware capabilities. Towards this direction, HW-NAS for transformers [11], [12], reduction of attention span [27], [28], and sparse transformers [29] have been presented.

A. HARDWARE-AWARE NEURAL ARCHITECTURE SEARCH

HW-NAS extends traditional NAS by incorporating hardware constraints into the search process. This approach ensures that the resulting model not only achieves high performance in terms of accuracy but is also optimized for energy efficiency, latency, and memory usage, which are all critical for the edge deployment [30]. We may benefit from HW-NAS to deploy language models on edge, as small language models (SLMs), so that no communication to cloud is needed. HW-NAS can significantly reduce the inference time of SLMs, improving the responsiveness of AI applications on edge devices.

The Evolved Transformer [11] uses evolutionary block-based NAS to search for a more efficient transformer architecture. The proposed search space uses two branch blocks to search for viable architectures. The branches allow for either adjacent operations or skip connections. However, these skip connections are limited to only skipping a single block, in contrast to our search, which allows skip connections to travel any number of layers. A decoder only variation of The Evolved Transformer showed promising results on Billion Word Benchmark [31] but little experimentation was completed on other language modeling benchmarks.

Hardware-Aware Transformer (HAT) [12] uses a weight-sharing supernet to search for hardware-efficient Transformer translation models, targeting a variety of hardware types (CPU, GPU, and IoT device). A SuperTransformer is trained and smaller edge-friendly sub-models are sampled from the supernet. HAT archives a 2.7 × speedup and 3.6 × smaller size over Evolved Transformer with 12,041 × less search cost and no performance loss. However, HAT only targets translation models, and the search space focuses on tuning model hyperparameters rather than searching for unique architectures.

AutoDistil [32] searches for optimal student models by distilling knowledge from a BERT teacher model. The search trains a task-agnotisic SuperLM to search for four hyperparameters in the transformer blocks. AutoDistil achieved fewer parameters over previously proposed knowledge distillation methods such as DistilBERT [33] while maintaining comparable accuracies on GLUE benchmark.

Our proposed NAS aims to target the shortcomings of previous methods. We use a chain-based search space to allow for architectures to vary through their depth. Chain-based search space also allows for skip connection to travel any distance in a model.

B. EFFICIENT TRANSFORMERS

Transformer provides state-of-art-results in NLP but is limited in its deployment by its $\mathcal{O}(N^2)$ complexity, where N is the length of the input sequence. Previous works have proposed numerous methods to reduce the computational complexity of transformer's self-attention mechanism [27], [28], [29], [34], [35], [36].

Adaptive attention span [27] employs a learnable attention span of the key matrix K to reduce the size of the $QK^\intercal$ computation (refer to Figure 4(a)). A gradient-based masking function is applied to the key array to allow the attention span to be learnable via gradient descent during the model's training. Adaptive attention span achieves state-of-the-art results with a sequence input length of 8,000 on text8 and enwiki8, but smaller input sequence lengths were not considered. Longformer [28] uses a self-attention window method to reduce the complexity of self-attention. The window sizes are manually set using the findings from [27] and are not learned to find their optimal size.

Luna [36] reduces the computational complexity of transformer's attention mechanism to linear complexity. Luna employs two nested linear attention functions to approximate the softmax function. The effectiveness of Luna is on par or better in masked language modeling benchmarks. Sparsity has been shown to be another effective method for reducing complexity of transformer, while also benefiting the scalability of transformer models [29]. Sparse variants of transformers are able to perform on the same accuracy level of a fully dense transformer with multiple times of speed up.

We propose learnable attention window sizes that are learned by during training, building upon previous works [27]. To benefit from real-world speedups, we introduce a custom CUDA kernel for our learnable attention window.

IV. PROPOSED SEARCH METHODOLOGY

A. MULTI-STAGE EVOLUTIONARY NAS ALGORITHM

In this work, we propose a multi-stage evolutionary algorithm, as a search strategy for NAS, to efficiently navigate a large and flexible search space whose scope extends beyond a single block as shown in Figure 2. In addition, our search directly communicates with resource-constrained edge devices to obtain the actual latency for inference. We aim to answer two questions when searching for efficient language models; is an attention layer necessary at every block? and can long skip connections convey low-level embedding features to the end of the model to benefit the model's performance?

Prior work on searching efficient transformer architectures limit the search space to a single block [11]. More specifically, the Evolved Transformer allows searching a block having two operator branches [11]. The authors of HAT [12] train a SuperTransformer that allows elastic embedding / hidden dimensions to change within a block without altering the block structure. Our search space aims to provide little restriction on what architectures can be explored, with the aim of discovering unique yet efficient transformer architectures with long skip connections.

The proposed multi-stage evolutionary NAS is designed to regularize the search and encourage further exploration of the search space. Our evolutionary algorithm can be broken down into three stages (i.e., multi-stage evolution in Figure 2); ① aggressive seeding, ② fine-tuning, and ③ mass extinction. These stages are repeated, with each repetition being called a *session* in this article.

① **Aggressive seeding** aims to create a large diversity in the initial population by mutating a randomly selected parent base model multiple times. The number of mutations applied to each new child model is selected at random between a range of n_A to $2n_A$, we set n_A to 20. The five mutations are depicted in Figure 2 and described in detail in Section IV-C. For the first session of the search, child models are seeded from a known successful base model, i.e., GPT-2 Small [3]. In the latter sessions, the base models consist of the surviving

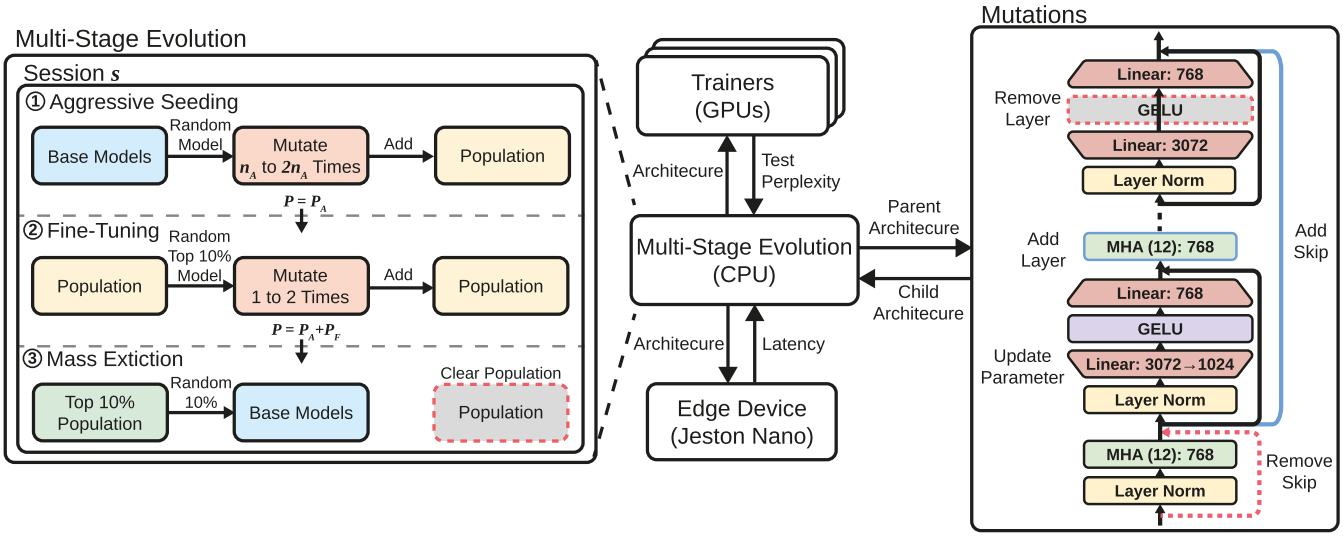

FIGURE 2. **Overview of the proposed multi-stage evolutionary algorithm for the extensive neural architecture search for edge-friendly Skipformers. Edge devices (NVIDIA Jetson Nano) are connected to GPU servers to provide real-time latency estimates.**

models from all previous sessions. Aggressive seeding is repeated until the population reaches a size of P_A, which we set to 100.

② **Fine-tuning** allows for promising architectures to be further developed by gradually mutating them. A single model is selected randomly from the top 10% of the fittest models in the population, e.g., a single model is randomly selected from the top 22 models from a population size of 220. The selected model has $n_F = 1$ or 2 mutations randomly applied, then is added to the population. Fine-tuning is repeated until the population reaches a size of $P_A + P_F$, which we set P_F to 200.

③ **Mass extinction** regularizes the search by preventing the search focusing too deeply in one direction and encourages exploration of other directions. Mass extinction achieves regulation by removing most models from the population, providing a clean start for the next population to generate novel architectures. From the top 10% of models, 1% of population size are randomly selected from the top 10% population to be added to the pool of base, e.g. 3 models randomly sampled from 30 top models. Finally, the population is purged by removing all models, leaving an empty population for the next session.

B. PROGRESSIVE FITNESS SCORE

Our multi-stage NAS uses a progressive fitness score to enable our NAS to be latency-aware. The objective of the proposed progressive fitness score is to find efficient language modeling architectures that enables real-time language processing on the edge. Thus, we consider two metrics in our fitness score, i.e., the model's test perplexity and its inference latency on the edge device. Perplexity is calculated from the exponential of the cross-entropy loss between the ground truth and the model's prediction.

Our fitness score utilizes targets on both perplexity and inference latency. The latency target 'l_T' rewards models that are faster than the target and penalizes models that are slower than the target. The test perplexity target 'p_T' progressively shrinks, allowing the early stages of the search to be more flexible, while ensuring that the latter sessions of the search are more defined. The fitness score initially allows for a target of within 10% of the target perplexity and shrinks the acceptable perplexity by 1% for each session. The final perplexity target encourages the fittest models to be at least equal or better than the target. We set the target perplexity and target latency from the seeding base model. The progressive fitness score is defined as:

$$F_m = \max\left(\frac{p_m}{(1.1 - 0.01s)\cdot p_T}, 1\right)\cdot\left(\frac{l_m}{l_T}\right)^{\lambda}, \qquad (1)$$

where p_m is the test perplexity of the current model m, l_m is the inference latency of the model, s is the session index, p_T is the target perplexity, and l_T is the target latency. The λ is set to 0.1 which controls the relative importance of the latency to the perplexity of the model. Lower λ values will prioritize the model perplexity, whilst higher λ values with prioritize the model latency.

C. SEARCH SPACE

Previous methods use block-level search to reduce the overall search space of computationally expensive NAS [37]. The block-level search only changes layer parameters within the block, i.e., number of attention heads, hidden dimensions for linear layers, and embedding dimensions [12]. While these methods are effective at reducing the search time, they restrict the flexibility of the search.

In our search space, we discard the use of blocks to allow for a greater search space to be explored, promoting the discovery of block-free novel architectures. This strategy is

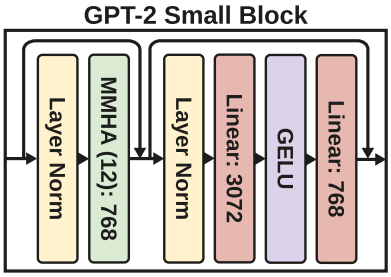

FIGURE 3. Original GPT-2 Small block, repeated 12 times in full model.

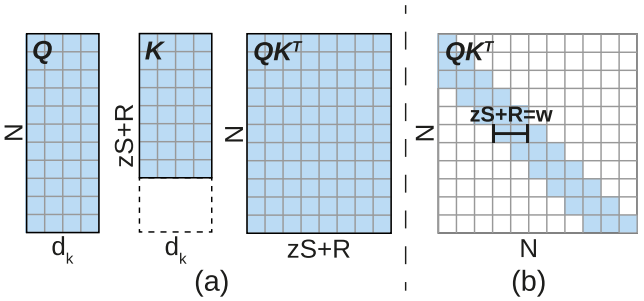

FIGURE 4. (a) Adaptive attention span [27]. (b) Proposed learnable attention window applied to $QK^\intercal$ output (white regions are zeros).

possible for small language models since the scalability is less of a concern compared to the server-scale LLM. Each architecture is represented as a chain-based layer list instead of dividing the model into identical blocks as per the original GPT-2 Small block (Figure 3). Our unrestricted chain-based search space allows novel unexplored architectures to be discovered. An architecture that can vary throughout its depth may provide models that are better optimized for the task than block-based search spaces. However, using an unrestricted search space requires more search time and more computational energy. We aim to reduce search time by using early stopping in model training (Section VII-A).

The search includes five mutations that have equal chance of selection; add/remove skip connection, add/remove layer, and update layer hyperparameter. A flexible embedding dimension offers the opportunity for the embedding dimension to vary throughout the depth of the model.

Add/remove skip connection randomly selects a starting layer and ending layer within the model for adding a skip connection. The rationale behind this is to let skip connection transverse any number of layers within the architecture so that it can pass information outside the original transformer block design. When removing a skip connection, a skip connection is randomly selected from the model.

Add/remove layer randomly selects a location to add a layer and a layer type from the following: linear, normalization, multi-head attention (MHA) or non-linear activation. Linear/MHA layers have the input/embedding dimension set to the output dimension of the last layer before the position of the new layer. After the addition of a linear layer, there is a 50% chance a non-linear activation layer to be added after. Non-linear activation function include ReLU, GELU [38], and squared ReLU [39]. Additionally, we add a dropout layer after the new layer if there is no dropout after the placement of the new layer. When a layer is added, layer dimensions are checked throughout the model to ensure dimensional consistency.

Update layer hyperparameter randomly selects a linear or MHA layer in the model, then randomly updates one of its hyperparameters. For linear layers, output dimension can be randomly updated to any value that is divisible by 192 between 192 and 4,416. The subsequent linear layer has its input dimension set to the new output dimension of the updated layer. For MHA layers, the number of heads is randomly updated from 4, 8 or 12.

D. LEARNABLE ATTENTION WINDOW IN LANGUAGE MODELS

Transformer owes its success to the self-attention mechanism. However, the self-attention mechanism is limited by its squared complexity in time and memory, $\mathcal{O}(N^2)$ where N is the length of the input sequence. The $\mathcal{O}(N^2)$ complexity is a result of the $QK^\intercal$ operation, where Q and K have a size of $\mathbb{R}^{N\times d_{model}}$, where d_{model} is the embedding dimension. Attention between all tokens has been shown to not be always required [27], [28], [40].

We propose a learnable attention window for language models, a learnable sliding window to reduce the number of computations of the self-attention layer in language models. Unlike adaptive attention span, where they learn the span of the key and value matrices (Figure 4a), our method learns the optimal self-attention window using gradient descent during training (Figure 4b). Our method only completes necessary computations and also avoids calculating values that are masked by the attention mask found in causal language modeling, reducing complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(wN)$, where $w = zS + R$ is the attention window size. A custom developed CUDA-accelerated kernel enables speed-up on CUDA GPUs.

For the window size to be learnable by gradient descent via backpropagation, a differentiable masking function must be introduced into the self-attention mechanism. We use a masking function that uses *tanh*-based ramp function, as shown in Figure 5 (blue solid line). Our *tanh* ramp function is capable of producing smaller attention windows than the linear ramp function used in [27], while maintaining the same or better perplexity (see Table 3 in Section VII-B).

To train the attention window, we first initialize a base mask m_b of dimension $\mathbb{R}^{N\times N}$. Each element of the base mask is initialized as follows.

$$m_{b[i,j]} = -|i-j|\,. \tag{2}$$

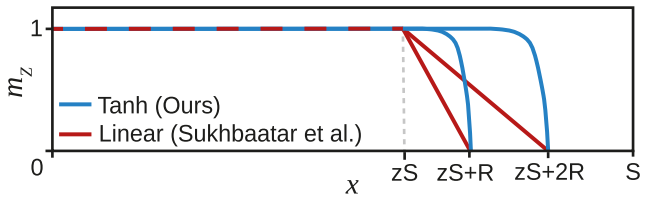

FIGURE 5. Tested masking functions for training self-attention windows: our *tanh* function (blue) and a linear function (red; Sukhbaatar et al. [27]).

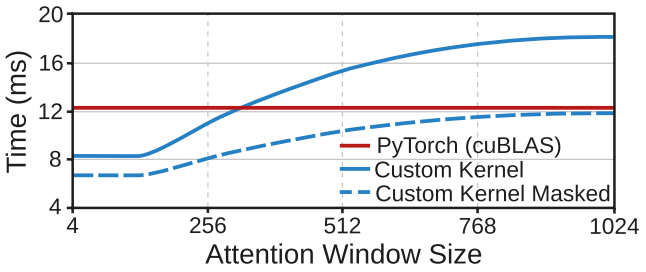

FIGURE 6. Our custom attention window CUDA kernel (blue) compared to PyTorch *torch.matmul* (red). Attention window varies from 4 to 1024 with batch size of 1, number of heads 12, and sequence length of 1024.

A differentiable attention window mask is updated at each forward pass with the following ramp equation.

$$m_w = \max\left[0, \tanh\left(\frac{4}{R}\left(m_b + zS + R\right)\right)\right], \tag{3}$$

where R is the minimum ramp size, S is the maximum window, m_b is the base mask, and z is the learnable parameter which controls the window size. We can describe z as adjusting the percentage of the maximum window, z must be minimized to reduce the attention window. Finally, the mask is element-wise multiplied with the attention output, i.e. $(QK^\intercal) \odot m_w$.

At the beginning of training, the attention window is at its full size, by initializing all z values to 1. The attention window is encouraged to shrink during training via the addition of an attention window loss to the model's loss.

$$l_w = \sum_{i=0}^{n} \gamma S z_i, \tag{4}$$

where γ is the loss coefficient and i is the layer index.

E. CUDA KERNEL FOR NARROWED ATTENTION WINDOW

We implemented a custom CUDA kernel for the narrowed attention window that enables real speed-up over the cuBLAS-based `torch.matmul` function. The kernel archives computational speed-ups by only calculating values inside the attention window at the $QK^\intercal$ output matrix. The kernel also supports the masked attention found in causal language modeling and will ignore computation for forward tokens, further reducing computation. With an input sequence length N of 1,024 we achieve up to a 1.9x speed up on the target NVIDIA Jetson device over full attention (Figure 6). Due to the CUDA thread and block allocation, there in no speed up below an attention window size of 128.

V. EXPERIMENTAL SETUP

A. TRAINING LANGUAGE MODELS DURING SEARCH

Our language model search focuses on causal language modeling, where the probability of the next token in the sequence is predicted. WikiText103 [41] is used for training and testing of architectures during the search, and OpenWebText [42] for full training and testing of the searched architectures in the final results. A vocabulary size of 50,257 tokens is used for all datasets, with input sequence length of 1,024 tokens. We preprocess the sequences and encode tokens using the same methods described in [3]. Sequences are packed together to create sequence lengths of 1,024. If the sequence is too long, we split the sentence and start the next input with the second half of the split sequence. During the search, a batch size of 5 is used to enable training on a single GPU. Final results are trained for 2 epochs on OpenWebText with a batch size of 48, resulting in 49,152 tokens per batch. Penn Treebank [43] and enwik8[1] are used as zero-shot benchmarks. Datasets are divided into train, test, and validation splits using the splits provided in the Hugging Face dataset hub.[2,3,4,5]

B. NATURAL LANGUAGE UNDERSTANDING BENCHMARK

To evaluate each model's generalizability and performance, we use GLUE benchmark [44]. GLUE benchmark comprises nine datasets that cover a variety of NLU tasks. The following datasets are included; **CoLA** [45] tests for grammatical acceptability, **SST-2** [46] predicts sentiment of movie reviews, **MPRC** [47] assesses semantic equivalence between sentence pairs, **QQP**[6] determines if question pairs are paraphrases of each other, **STS-B** [48] measures similarity between sentence pairs, **MNLI** [49] evaluates textual entailment across ten different genres of text, **QNLI** [50] evaluates semantic textual similarity on question-answer pairs, **RTE** [51] finds textual entailment between sentence pairs, **WNLI** [52] tests if a sentence with the pronoun substituted is entailed by the original sentence. Datasets are divided into train, test, and validation splits using the splits provided in the Hugging Face dataset hub[7] and organized into sequence lengths of 128. We fine-tune the pre-trained models for 5 epochs on each dataset, then take the test accuracy.

C. LATENCY COLLECTION

We collect actual latency measurements directly from target hardware rather than using a latency predictor, as there

[1] http://mattmahoney.net/dc/textdata.html
[2] https://huggingface.co/datasets/enwik8
[3] https://huggingface.co/datasets/openwebtext
[4] https://huggingface.co/datasets/ptb_text_only
[5] https://huggingface.co/datasets/wikitext
[6] https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs
[7] https://huggingface.co/datasets/glue

is sufficient time to collect each metric accurately due to the slower nature of evolutionary NAS [53] compared to supernet-based [9], [12] and zero-shot [13] NAS methods. The forward pass of the model is measured over 400 passes with a fixed input length of the maximum sequence length of 1,024. Outliers outside 3σ of the mean are removed before calculating the final mean of the model's latency. All latencies are reported in seconds, unless stated otherwise.

D. TRAINING SETUP AND HYPERPARAMETERS

1) TRAINING HYPERPARAMETERS

We complete all training using Adam optimizer with a learning rate of 1e-4, betas of (0.9, 0.98) and epsilon of 1e-9. We set the window loss coefficient γ in Eq. 4 to 1e-5.

2) SEARCH HYPERPARAMETERS

For search hyperparameters, we set P_A to 100 and P_F to 200, giving a total population P of 300 per session. We search up to session 10 which totals 3,000 searched architectures. The first seeding base model is set to GPT-2 Small for the search. In Eq. 4, p_T and l_T are set to GPT-2 Small's test perplexity and latency, respectively. The fitness score coefficient λ is set to 0.1 for all searches. Early stopping at 100,000 steps is used to reduce the search time (Section VII-A).

3) HARDWARE SETUP

We use 40 NVIDIA GPUs to train and evaluate architectures for the search, ranging from Titan Xp up to A100. We measure actual latencies from two NVIDIA Jetson Nano 4GB boards in 10W power mode.

VI. EXPERIMENTAL RESULTS

A. SEARCH RESULTS AND ARCHITECTURAL FEATURES

Out of the 3,000 searched models, we select two models for testing, i.e., Skipformer A (Figure 7), and Skipformer B. Skipformer A has the best final fitness score, and Skipformer B is a model that produces slightly worse perplexity results but greater improvement in latency.

In both Skipformer models, the total number of MHA and linear layers are reduced from the original GPT2-Small model. Skipformer A features a reduction of 2 MHA and 3 linear layers, and Skipformer B features a reduction of 3 MHA and 5 linear layers. Skipformers also vary greatly in architecture compared to GPT2-Small. In the searched models, many of the remaining linear layer dimension have been reduced from the original dimension of 3,072. The largest reduction is found in the first linear layer of Skipformer A with an output dimension of 384. Little to no change was observed in the number of heads per MHA layer. These three unique architectural features observed in the search reduce the total number of parameters in the model, boosting on-device efficiency.

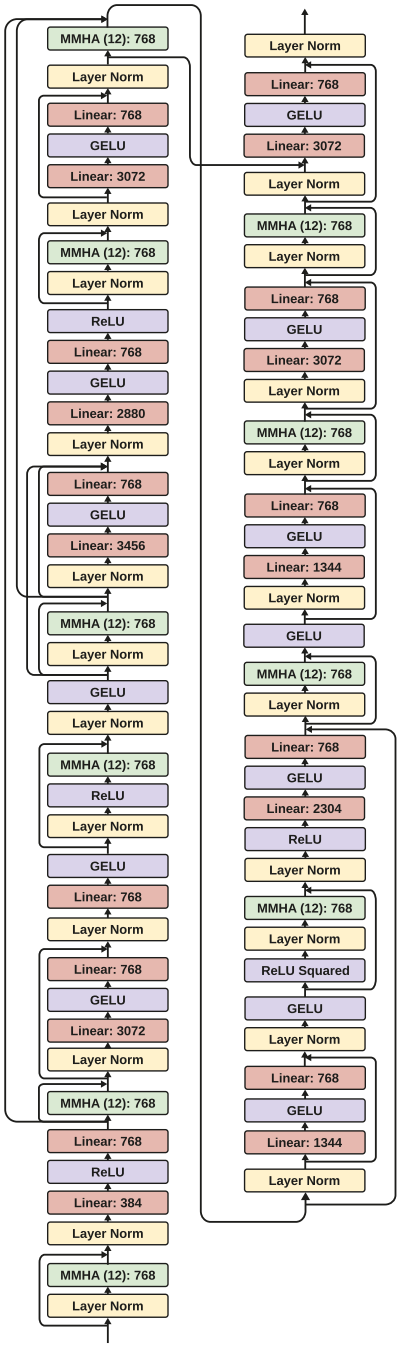

FIGURE 7. Skipformer A architecture.

1) NON-REPEATABLE ARCHITECTURAL PATTERNS

The searched models vary through the depth of the model. This feature may not have been discovered using a repeatable block-based search algorithm. Block-based search algorithms repeat the searched block multiple times to create the full model architecture, creating a repeatable pattern. Although some structural similarities to GPT-2 Small remain, there is greater variation in the architectures. It is found that the earlier layers in an architecture have many variations, while

TABLE 1. All models are pre-trained on OpenWebText for two training epochs. All results are reported as perplexity on the respective test datasets, excluding enwik8, which is reported as bit per character. Latencies are actually measured on NVIDIA Jetson Nano. Lower is better.

Model	OpenWebText	Billion Word	Penn Treebank	WikiText-103	enwik8	Parameters	Latency (s)
GPT-2 Small	23.84	77.26	88.75	45.91	36.48	118.7M	3.17
GPT-2 Small-W	24.65	75.74	102.04	52.30	33.42	118.7M	2.95
Skipformer A	28.21	84.46	112.99	59.33	39.34	95.9M	2.44
Skipformer B	31.29	104.41	151.61	73.28	45.26	85.2M	2.17

the latter layers follow patterns found in the original GPT-2 Small block.

2) COUNTERINTUITIVE LAYER ORDERS

In the searched architectures, we can see combinations of layers that would not usually be considered when manually designing a model. Some of these features include, double non-linear activations, i.e. ReLU Squared → GELU, and single linear layer with same input and output dimensions.

3) LONG SKIP CONNECTIONS

Within each Skipformer there are skip connections that transverse multiple layers. It is believed that these connections reintroduce lost information back into the final layer of the model.

B. PERFORMANCE ON LANGUAGE MODELING

To evaluate the quality of Skipformers, we evaluated the performance of each model on five datasets, i.e., OpenWebText, Billion Word, Penn Treebank, WikiText-103, and enwik8, and summarized in Table 1. We take GPT-2 Small as the baseline as an on-device language model. GPT-2 Small with reduced attention window was assessed as a faster baseline (GPT-2 Small-W). For all models, pre-training was performed with 2 epochs on OpenWebText. The metrics for other datasets shown in Table 1 are zero-shot. Skipformer A reduces the model size by 19.2% and the latency by 23.3% compared to GPT-2 Small. With two training epochs, Skipformer A degrades the accuracy by 19.1% and 12.6% on average compared to GPT-2 Small and GPT-2 Small-W, respectively. Skipformer B is significantly worse in accuracy while reducing model size and latency by another 9.0% and 8.2%, respectively.

C. FINE-TUNING TO OTHER LANGUAGE TASKS

Accuracy loss in large-scale datasets in Table 1 does not necessarily mean that the searched models will perform poorly in smaller NLU datasets. We believe that the GLUE benchmark, i.e., smaller NLU datasets, is more suitable for evaluating the quality of on-device language models. Thus, we fine-tune the searched models for the GLUE benchmark and the results are reported in Table 2. Fine-tuning on each dataset is completed after 2 pre-training epochs on the OpenWebText. As summarized in Table 2 and Figure 8, the accuracy of Skipformer A on GLUE benchmark is similar to that of GPT-2 Small or GPT-2 Small-W (-0.95% on average).

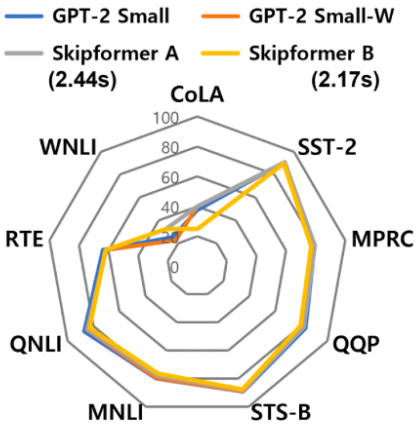

FIGURE 8. Overview of GLUE results for the evaluated language models.

Even the smallest and fastest Skipformer B loses accuracy only by 2.6% on average, while improving the latency by 31.5%.

D. EFFICIENCY OF OUR CUSTOM CUDA KERNEL

When considering the speed-up of the original attention method, our learnable attention window with the custom CUDA kernel can achieve a speed-up of up to 1.9×. Even on the original GPT-2 Small model, we can see a 6.9% speed up, when comparing GPT-2 Small with GPT-2 Small-W in Table 1, simply by allowing the learnable attention window with our custom CUDA kernel.

VII. ABLATION STUDIES

A. ABLATION STUDY ON EARLY STOPPING

We employ early stopping in our proposed search method to reduce the training time of each model, in turn reducing the overall search time. We first completed a study of how effective early stopping is for reducing search time whilst still providing accurate final training accuracies. We fully trained 300 models to 500,000 steps on WikiText-103 (one session size), while recording the test perplexity at every 10,000 steps. Using this data, the test perplexity ranking can be calculated relative to the other models at each interval. With Spearman's rank correlation, we can find the ranking correlation at interval compared to the final test perplexity ranking. It was found that over 100,000 training steps there is a strong collection > 0.8 for the test accuracy ranking of the model in the population (Figure 9).

TABLE 2. Performance on GLUE task test set (trained for 5 epochs on each dataset). CoLA is reported with Matthews correlation coefficient. MRPC and QQP are reported with accuracy and F1. STS-B is reported with Pearson and Spearman correlation coefficient. MNLI is reported with accuracy on the matched and mismatched test sets. All other tasks are reported with accuracy. All values have been scaled by 100. Higher is better.

Model	CoLA	SST-2	MPRC	QQP	STS-B	MNLI	QNLI	RTE	WNLI	Mean
GPT-2 Small	37.57	90.83	79.41/86.36	83.20/83.03	89.28/85.94	79.96/80.18	86.95	63.90	25.35	74.77
GPT-2 Small-W	40.04	89.91	79.66/85.96	79.88/79.90	88.93/85.46	79.65/79.63	85.01	62.09	22.54	73.74
Skipformer A	40.07	90.60	79.41/86.32	81.88/81.47	88.42/84.78	77.92/78.72	84.44	61.73	32.39	74.47
Skipformer B	24.97	89.68	76.96/84.49	79.47/79.42	87.69/84.13	76.56/77.92	82.85	62.09	32.39	72.20

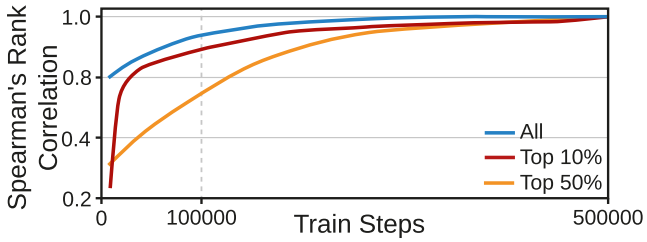

FIGURE 9. Spearman's rank correlation of test perplexity against test perplexity at 500,000 steps when calculated at 10,000 step intervals with a sample size of 300. A strong rank correlation can be seen at 100,000 steps in both all models and top 10% of models.

TABLE 3. Comparison on the quality of learnable attention window using different ramp functions. Results are extracted by training GPT-2 Small on OpenWebText for one epoch.

Ramp Function	Test Perplexity	Min Window	Mean Window	Max Window
None	43.21	16	16	16
Linear	28.77	16	56	342
Tanh	28.26	16	52	333
Full	27.46	1024	1024	1024

B. ABLATION STUDY ON ADAPTIVE ATTENTION WINDOW MASKING FUNCTION

To understand how the masking function affects the model's accuracy and latency, we performed a set of experiments in finding the optimal masking function (Table 3). We trained GPT-2 Small with learnable attention window for 1 epoch on OpenWebText with no ramp, *tanh* ramp, and *linear* ramp functions. With no ramp function, the window size continuously decreases to its minimum size, causing poor test perplexity. The study indicates that a ramp function is required to learn the attention window effectively, and our proposed *tanh* ramp function outperforms *linear* ramp function on test perplexity and achieves smaller window sizes.

Training GPT-2 Small with our learnable attention window, i.e., GPT-2 Small-W, showed that full self-attention is not necessary for strong perplexity (Table 1). We find that most of the learned attention windows converge to their minimum learnable size, which is set to 16 in our experiments (Figure 10).

VIII. DISCUSSIONS

The main limitation of our proposed method is the search time, running for ∼18,000 GPU hours. Our search was completed using on a selected edge device, i.e., NVIDIA Jetson Nano, but our proposed method and code are designed so that target devices can be easily changed, ranging from server-scale GPU cards to edge devices such as Raspberry Pi. In order to target other devices, the initial base model must run on the target hardware to collect the target latencies for the NAS fitness score.

Self-attention acceleration is only provided for CUDA-supported devices. Self-attention acceleration must be developed for the target hardware when targeting devices outside of CUDA enabled hardware. Further hardware optimization must be considered to improve efficiency of all layer types within the searched architectures. Our proposed method can lack scalability as each architecture is unique and features no repeatable structures. Thus, we may extract important features of Skipformer per stage and establish a scalable architecture.

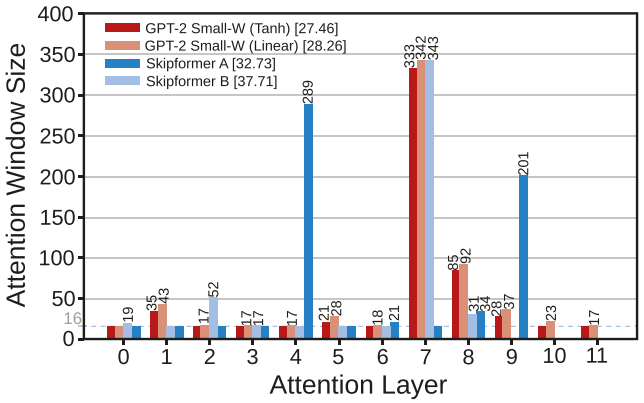

FIGURE 10. Attention window size per attention layer. All models are trained for 1 epoch on OpenWebText. Unlabeled bars have value of 16. Value in bracket denotes the test perplexity on OpenWebText.

IX. CONCLUSION

Our proposed multi-stage latency-aware evolutionary NAS produced novel transformer architectures to support on-device language modeling. In addition, our learnable self-attention window reduced the computation cost of language models using our custom CUDA kernel. By applying the proposed learnable attention window on GPT-2 Small, we could achieve 7% speed up (without any architectural modification). The searched model with the learned window size has shown promising results on GLUE benchmark

comparable to the original GPT-2 Small baseline model. Skipformer achieves 23.3% speed up and requires 19.2% less memory with negligible accuracy loss on GLUE benchmark compared to the GPT-2 Small. As a future work, shallower models should be explored by adjusting the search space. Furthermore, custom hardware can be designed to further improve the speed-up of the restricted attention window.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, Long Beach, CA, USA: Curran Associates, 2017.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. NAACL*, vol. 1, Jun. 2019, pp. 4171–4186.

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[4] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, no. 140, pp. 1–67, 2020.

[5] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Aging evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 2, 2019, p. 2.

[6] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1294–1303.

[7] B. Chen, P. Li, B. Li, C. Lin, C. Li, M. Sun, J. Yan, and W. Ouyang, "BN-NAS: Neural architecture search with batch normalization," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, 2021, pp. 307–316.

[8] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, and V. Vasudevan, "Searching for MobileNetv3," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1314–1324.

[9] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 439–451.

[10] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2815–2823.

[11] D. So, Q. Le, and C. Liang, "The evolved transformer," in *Proc. Int. Conf. Mach. Learn.*, vol. 97, 2019, pp. 5877–5886.

[12] 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 Annu. Meeting Assoc. Comput. Linguistics*, 2020, pp. 7675–7688.

[13] M. Javaheripi, S. Shah, S. Mukherjee, T. L. Religa, C. C. Mendes, G. H. de Rosa, S. Bubeck, F. Koushanfar, and D. Dey, "LitetransFormerSearch: Training-free on-device search for efficient autoregressive language models," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2022, pp. 24254–24267.

[14] D. Jurafsky and J. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Upper Saddle River, NJ, USA: Prentice-Hall, 2008.

[15] C. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. Cambridge, MA, USA: MIT Press, 1999.

[16] Y. Goldberg, "A primer on neural network models for natural language processing," *J. Artif. Intell. Res.*, vol. 57, pp. 345–420, Nov. 2016.

[17] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Comput. Intell. Mag.*, vol. 13, no. 3, pp. 55–75, Aug. 2018.

[18] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, and A. Askell, "Language models are few-shot learners," in *Proc. NeurIPS*, 2020, pp. 1877–1901.

[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[20] A. Radford. *Improving Language Understanding With Unsupervised Learning*. OpenAI. Accessed: Jun. 28, 2024. [Online]. Available: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[21] K. Ethayarajh, "Understanding the learned representations of transformers for cross-lingual transfer," 2019, *arXiv:1909.01500*.

[22] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," *J. Mach. Learn. Res.*, vol. 3, pp. 1137–1155, Feb. 2000.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, and A. N. Gomez, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.

[24] B. Zoph and Q. Le, "Neural architecture search with reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 2841–2856. [Online]. Available: https://openreview.net/forum?id=r1Ue8Hcxg

[25] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 19–35.

[26] H.-I. Liu, M. Galindo, H. Xie, L.-K. Wong, H.-H. Shuai, Y.-H. Li, and W.-H. Cheng, "Lightweight deep learning for resource-constrained environments: A survey," *ACM Comput. Surv.*, vol. 56, no. 10, pp. 1–42, Oct. 2024.

[27] S. Sukhbaatar, E. Grave, P. Bojanowski, and A. Joulin, "Adaptive attention span in transformers," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 331–335.

[28] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," 2020, *arXiv:2004.05150*.

[29] S. Jaszczur, A. Chowdhery, A. Mohiuddin, L. Kaiser, W. Gajewski, H. Michalewski, and J. Kanerva, "Sparse is enough in scaling transformers," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 9895–9907.

[30] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 10726–10734.

[31] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson, "One billion word benchmark for measuring progress in statistical language modeling," 2013, *arXiv:1312.3005*.

[32] D. D. Xu, S. Mukherjee, X. Liu, D. Dey, W. Wang, X. Zhang, A. Awadallah, and J. Gao, "Few-shot task-agnostic neural architecture search for distilling large language models," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Red Hook, NY, USA: Curran Associates, 2022, pp. 28644–28656. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/b7c12689a89e98a61bcaa65285a41b7c-Paper-Conference.pdf

[33] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of BERT: Smaller, faster, cheaper and lighter," in *Proc. 5th Workshop Energy Efficient Mach. Learn. Cognit. Comput., NeurIPS*, 2019.

[34] N. Kitaev, L. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–12.

[35] A. Roy, M. Saffar, A. Vaswani, and D. Grangier, "Efficient content-based sparse attention with routing transformers," *Trans. Assoc. Comput. Linguistics*, vol. 9, pp. 53–68, Feb. 2021.

[36] X. Ma, X. Kong, S. Wang, C. Zhou, J. May, H. Ma, and L. Zettlemoyer, "Luna: Linear unified nested attention," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, Dec. 2021, pp. 2441–2453.

[37] J. Fang, Y. Sun, Q. Zhang, Y. Li, W. Liu, and X. Wang, "Densely connected search space for more flexible neural architecture search," in *Proc. CVPR*, 2020, pp. 10628–10637.

[38] D. Hendrycks and K. Gimpel, "Bridging nonlinearities and stochastic regularizers with Gaussian error linear units," 2016, *arXiv:1606.08415*.

[39] D. So, W. Mańke, H. Liu, Z. Dai, N. Shazeer, and Q. V. Le, "Searching for efficient transformers for language modeling," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 34, 2021, pp. 6010–6022.

[40] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, and L. Yang, "Big bird: Transformers for longer sequences," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 33, 2020.

[41] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer Sentinel mixture models," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1851–1865. [Online]. Available: https://openreview.net/forum?id=Byj72udxe

[42] A. Gokaslan and V. Cohen. (2019). *OpenWebText Corpus*. [Online]. Available: http://Skylion007.github.io/OpenWebTextCorpus

[43] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of English: The Penn Treebank," *Comput. Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[44] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–20.
[45] A. Warstadt, A. Singh, and S. R. Bowman, "Neural network acceptability judgments," 2018, *arXiv:1805.12471*.
[46] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proc. EMNLP*, vol. 1631, Jan. 2013, pp. 1631–1642.
[47] W. B. Dolan and C. Brockett, "Automatically constructing a corpus of sentential paraphrases," in *Proc. Int. Workshop Paraphrasing*, 2005, pp. 9–16.
[48] D. Cer, M. Diab, E. Agirre, I. Lopez-Gazpio, and L. Specia, "SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation," in *Proc. 11th Int. Workshop Semantic Eval. (SemEval)*, S. Bethard, M. Carpuat, M. Apidianaki, S. M. Mohammad, D. Cer, and D. Jurgens, Eds., Vancouver, BC, Canada: Association for Computational Linguistics, Aug. 2017, pp. 1–14, doi: 10.18653/v1/S17-2001. [Online]. Available: https://aclanthology.org/S17-2001
[49] A. Williams, N. Nangia, and S. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Human Lang. Technol.*, 2018, pp. 1112–1122. [Online]. Available: http://aclweb.org/anthology/N18-1101
[50] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "SQuAD: 100,000+ questions for machine comprehension of text," in *Proc. EMNLP*. Stroudsburg, PA, USA. Association for Computational Linguistics, 2016, pp. 2383–2392.
[51] I. Dagan, O. Glickman, and B. Magnini, "The PASCAL recognising textual entailment challenge," in *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, J. Quiñonero-Candela, I. Dagan, B. Magnini, and F. d'Alché Buc, Eds. Berlin, Germany: Springer, 2006, pp. 177–190.
[52] H. J. Levesque, E. Davis, and L. Morgenstern, "The Winograd schema challenge," in *Proc. AAAI Spring Symp., Log. Formalizations Commonsense Reasoning*, vol. 46, 2011, p. 47.
[53] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 4780–4789.

MATTHEW BODENHAM (Graduate Student Member, IEEE) received the M.Sc. degree in aerospace engineering from Loughborough University, U.K., in 2018, and the M.Sc. degree in data science from the University of Bath, U.K., in 2020. He is currently pursuing the Ph.D. degree with the Department of Electrical Engineering and Computer Science, Daegu Gyeongbuk Institute of Science and Technology (DGIST), South Korea.

His research involves optimizing deep leaning models for deployment on resource-limited edge devices. His research interests include neural architecture search and kernel optimizations for efficient deep learning on the edge.

JAEHA KUNG (Member, IEEE) received the B.S. degree in electrical engineering from Korea University, Seoul, South Korea, in 2010, the M.S. degree in electrical engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2012, and the Ph.D. degree in electrical and computer engineering from Georgia Institute of Technology, Atlanta, GA, USA, in 2017.

He is currently an Associate Professor with the School of Electrical Engineering, Korea University. Prior to joining Korea University, from 2017 to 2022, he was with the Department of Electrical Engineering and Computer Science, Daegu Gyeongbuk Institute of Science and Technology (DGIST), Daegu, South Korea. His current research interests include energy-efficient digital accelerators for deep learning, distributed learning systems, hardware design for artificial intelligence, and low-power VLSI design. He also serves on the Technical Committee for IEEE/ACM DAC, IEEE/ACM ISLPED, IEEE ISCAS, and IEEE AICAS.