HAMMER: Hardware-Friendly Approximate Computing for Self-Attention With Mean-Redistribution And Linearization

Seonho Lee, Ranggi Hwang, Jongse Park, and Minsoo Rhu

Abstract—The recent advancement of the natural language processing (NLP) models is the result of the ever-increasing model size and datasets. Most of these modern NLP models adopt the Transformer based model architecture, whose main bottleneck is exhibited in the self-attention mechanism. As the computation required for self-attention increases rapidly as the model size gets larger, self-attentions have been the main challenge for deploying NLP models. Consequently, there are several prior works which sought to address this bottleneck, but most of them suffer from significant design overheads and additional training requirements. In this work, we propose HAMMER, hardware-friendly approximate computing solution for self-attentions employing mean-redistribution and linearization, which effectively increases the performance of self-attention mechanism with low overheads. Compared to previous state-of-the-art self-attention accelerators, HAMMER improves performance by $1.2-1.6\times$ and energy efficiency by $1.2-1.5\times$.

Index Terms—Approximate computing, hardware accelerator, neural network, sparse computation, transformers

1 Introduction

In recent years, machine learning (ML) applications based on deep neural networks (DNNs) have become widespread. Along with the rapid growth of their applicability, natural language processing (NLP) is one of the most promising domains. Especially, large-scale pre-trained models (e.g., BERT, GPT, and MT-NLG) have demonstrated their efficacy for NLP advancement. These models adopt the `Self-attention` mechanism, which understands the subtle differences in meaning between different contexts. However, these advanced NLP models require huge computation and memory requirements with recent models suffering from even higher overheads due to their larger model architectures and their longer input sequences.

To mitigate this problem, there have been several prior works employing custom hardware-based acceleration for `Self-attention` via approximate computing (e.g. A^3 [1], ELSA [2], and SpAtten[3]). These works, however, suffer from the following two limitations. First, many of these prior studies [2], [3] incur additional training before deployment, which causes large overheads in the order of a few minutes to a few hours. Because this (re)training stage is required whenever the target application changes, such overheads cannot be easily amortized. Second, the implementation of approximation consumes non-negligible portion over the overall accelerator area (e.g., the approximation units take up more than 60% of the area for A^3 [1], and 50% for ELSA [2]), causing significant overheads.

To address the aforementioned challenges of prior approximation based `Self-attention` accelerators, we propose an algorithm-hardware co-design named *HAMMER*, a **H**ardware-friendly **A**pproxi**M**ate computing with **ME**an-**R**edistribution and linearization for `Self-attention` based NLP models. As detailed in Section 3, HAMMER requires a minimal extension to hardware resources for approximation while incurring much smaller computation overhead. To summarize our contributions:

1) We propose an efficient and hardware-friendly approximation scheme that does not require additional training for approximation. This makes our hardware design more versatile and easily accessible for deployment.
2) The proposed approximation algorithm demonstrates that the efficient and hardware-friendly approximation and pruning scheme can be adopted to the `Self-attention` mechanism with negligible impact on accuracy.
3) We present HAMMER, an accelerator for self-attention which is co-designed with the proposed approximation scheme. It outperforms the state-of-the-art `Self-attention` accelerators in terms of performance and area-efficiency.

2 Background and Motivation

The `Self-attention` mechanism is at the core of the transformer model. Because it computes the combinational relationships between inputs, the required computation scales quadratically with the length of the input [1], [2]. This quadratic scaling makes `Self-attention` very costly. Fig. 1 shows the latency breakdown of representative transformer-based NLP model BERT-Base profiled on a GPU system. The latency of `Self-attention` dominates as the sequence length increases. Because recent NLP models tend to adopt longer input sequences, accelerating `Self-attention` can significantly help improve end-to-end performance.

`Self-attention` mechanism can be generally formulated as in Eqs. (1), (2), and (3). It takes three matrices of inputs: $Query$, Key, and $Value$, with the dimension of (Number of Inputs)×(Hidden Dimension). [1], [8]

$$Score = Query \cdot Key^T \tag{1}$$

$$Prob = softmax(Score) \tag{2}$$

$$SelfAttention(Query, Key, Value) = Prob \cdot Value \tag{3}$$

$Prob$ in Eq. (2) reveals an interesting property as softmax function maximizes the differences between input values in $Score$ vector. We plotted the values of the elements in multiple rows of $Prob$ in Fig. 2 and observed that it follows a power-law distribution, i.e., most of the $Prob$ values are below 0.01. Considering that the range of softmax function is between 0 and 1, the values smaller than 0.01 have little impact on the final output of `Self-attention` mechanism, and we demonstrate this in Section 5. However, unlike its little impact on the final output, these unimportant values possess the majority portion (99%) in computation. Consequently, these unimportant near-zero values can potentially be *pruned out* to reduce a significant amount of computation while having a negligible impact on accuracy.

3 Algorithm

3.1 Pruning Unimportant Tokens

Based on our key observations in Fig. 2, we design a *sparse* version of $Prob$ as follows:

$$\widehat{Prob}_{ij} = \begin{cases} Prob_{ij} & \text{if } Prob_{ij} > threshold \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Because the softmax function makes $\widehat{Prob}$ very sparse, the computation required for Eq. (3) can be reduced with the help of sparse-dense

• *The authors are with the KAIST, Daejeon 34141, South Korea. E-mail: {seonho.lee, ranggi.hwang, mrhu}@kaist.ac.kr, jspark@casys.kaist.ac.kr.*

Manuscript received 22 October 2022; accepted 25 November 2022. Date of publication 4 January 2023; date of current version 30 January 2023.
This work was supported in part by the Super Computer Development Leading Program of the National Research Foundation of Korea (NRF) funded by the Korean government (Ministry of Science and ICT(MSIT)) under Grant 2020M3H6A1085498, in part by the National Research Foundation of Korea (NRF) funded by the Korea government (MSIT) under Grant NRF-2021R1A2C2091753, in part by the Institute of Information & communications Technology Planning & Evaluation (IITP) funded by the Korea government (MSIT) under Grant 2022-0-01037, Development of High Performance Processing-in-Memory Technology based on DRAM, and in part by Samsung Electronics Co., Ltd under Grant IO201210-07974-01. We also appreciate the support from the IC Design Education Center (IDEC) , Korea, for the EDA tools.
(Corresponding author: Minsoo Rhu.)
Digital Object Identifier no. 10.1109/LCA.2022.3233832

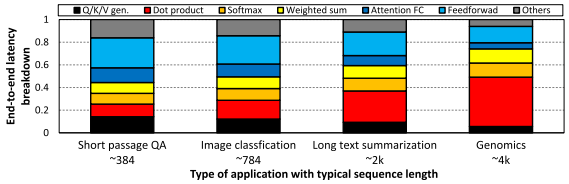

Fig. 1. End-to-end latency breakdown of BERT-Large based applications[4], [5], [6], [7]. Dot-product, softmax, and weighted sum comprise the `Self-attention` mechanism. Results are collected on NVIDIA A100.

matrix multiplication. Still, the overall computational complexity is bounded by the computation of Eq. (1), whose time complexity is $O(N^2)$. To tackle this problem, we can skip the computation for the element in $Score$ if the corresponding element in $\widehat{Prob}$ will be zero [1], [2]. Eq. (5) shows the modified version of Eq. (1). If an element in $Score$ of which the corresponding value in $Prob$ is less than the threshold, then instead of computing it, we mark it as negative infinity and continue the following computations. Then softmax maps negative infinity to zero, making the $\widehat{Prob}$ sparse as in Eq. (4). The main challenge here is that before we compute $\widehat{Score}$, we need to know in advance the $Prob$, which again requires $Score$. To address this problem, we present a hardware-friendly approximate scheme to compute the approximated version of $Score$ and predict the value of $Prob$ before computing Eq. (5). Following sections detail how we compute the approximation in terms of mean redistribution (Section 3.2) and linearization (Section 3.3)

$$\widehat{Score}_{ij} = \begin{cases} (Query \cdot Key^T)_{ij} & \text{if } Prob_{ij} > threshold \\ -inf & \text{otherwise} \end{cases} \quad i,j = 1....N \tag{5}$$

3.2 Attention Score Approximation

We apply static quantization scheme on the input $Query$ and Key values to approximate the computation of $Score$ (which is used for predicting $Prob$). A key obstacle here is that the polarity distribution of input values keep changing dynamically, even within a single inference (Fig. 3). This makes it challenging to choose a static reference for quantization. An alternative measure will be to employ K-mean clustering style *dynamic* reference quantization, but they generally incur heavy pre-processing and latency overheads. HAMMER addresses such challenge with a fast, lightweight static quantization with mean-redistribution which can adapt to dynamic input distribution at runtime.

Input Quantization. When computing the approximate $Score$ to predict the value of $Prob$, we use the aggressive quantization scheme for $Query$ and Key with 2-bit representation in Eq. (6). It divides the real number plane into four sections. If the absolute value of an input is larger than 4, it is mapped to 8 or -8 depending on its sign bit. For inputs of which the absolute value is smaller than 4, they are mapped to 1 or -1 in the same manner. We empirically chose the weight factor to 8 after carefully sweeping the factor from 1 to 32 in the power of 2. This quantization scheme can represent a wider range of values than simple binary quantization and can be implemented in hardware using simple shifters without an area-hungry multiplier

$$Quant(x) = \begin{cases} 8 \cdot sgn(x) & \text{if } |x| \geq 4 \\ sgn(x) & \text{otherwise} \end{cases} \tag{6}$$

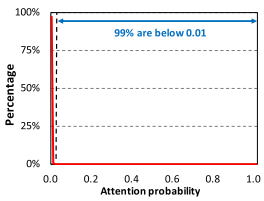

Fig. 2. Value distribution of $Prob$, in RoBERTa with a sample sequence length of 512.

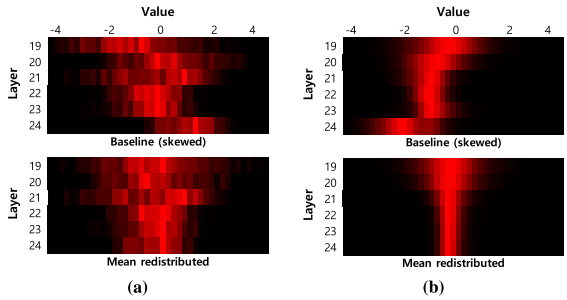

Fig. 3. Heatmap of the value distribution of (a) Key and (b) $Score$ in the first head of the last six layers in BERT-Large.

Mean redistribution. Fig. 3a shows the values in Key in the first head of the last six layers in the BERT-Large. The distribution of input values is usually skewed from zero, which makes the quantization result biased. To address this problem, we leverage the property of the softmax function that it is invariant to the constant offset in Eq. (7)

$$softmax(x_i) = \frac{exp(x_i)}{\sum_{j=1}^{N} exp(x_j)} = \frac{exp(x_i - c)}{\sum_{j=1}^{N} exp(x_j - c)}. \tag{7}$$

With this property, the mean values of each row in Key can be subtracted across the row dimension as shown in Eq. (8), minimizing quantization error without affecting the output value. It changes the value distribution of Key to be properly quantized. With this zero-centered Key, we can eventually have the equation to compute the approximate score as Eq. (9). Because the quantized values are all power of 2 (e.g. 1, -1, 8, -8), the approximated score can be computed through only addition and shift operations without any multiplications

$$(Mean_{Key})_j = (\sum_{i=1}^{N} Key_{ij})/N \tag{8}$$

$$Approx.\ Score = Quant(Query) \cdot Quant((Key - Mean_{Key})^T) \tag{9}$$

3.3 Softmax Linearization

There are several ways to implement the softmax function, such as lookup tables, fixed-point approximation, piecewise linear approximation, etc. However, depending on target precision, these implementations can suffer from the large size of the on-chip buffer, or require floating-point representation to cover a huge range of exponentials in softmax. Such implementation overheads can be relatively large for our target hardware-friendly lightweight approximation unit. Consequently, a key objective of our approach is to emulate only the core property of softmax with minimal hardware resources as detailed below.

Softmax can be viewed as a normalization function where an exponential function is applied to its input values (Eq. (7)). Instead of applying exponential before normalization to compute softmax, we may apply another alternative function that behaves similarly to exponential. We characterize the properties of the exponential core to the softmax as follows; i) Exponential is a positive function ii) Exponential is a monotonically increasing function.

ReLU function is the simplest candidate which mostly satisfies these two properties: its non-negative and monotonically increasing behavior for non-zero input values. In Fig. 4, we measured the accuracy of `Self-attention` for each of the aforementioned alternative functions to predict the value of $Prob$ and approximate the $Score$ as proposed in Section 3. EXP is the exponential function used in the original softmax function, so it's effectively an upper bound for the approximations. Not surprisingly, the quality of ReLU softmax is much worse than EXP, degrading sharply for a high pruning ratio.

The distribution of $Score$ which is the input of ReLU is mostly skewed as in Fig. 3b. For the distributions skewed to negative values, the behavior of ReLU, which cuts off negative values, causes lots of information to be lost. Fortunately, our proposed mean-redistribution approach described in Section 3.2 can mitigate this problem. Mean-

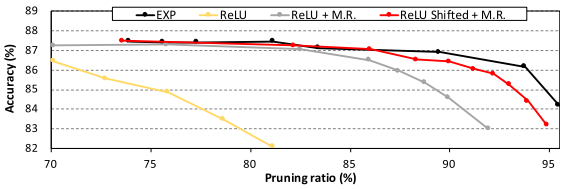

Fig. 4. Accuracy comparison on SQuADv2 between exponential and alternative functions with RoBERTa-Large.

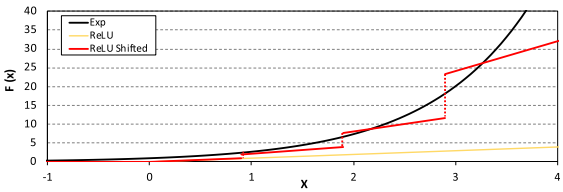

Fig. 5. Proposed alternative functions for exponential: ReLU and ReLU Shifted.

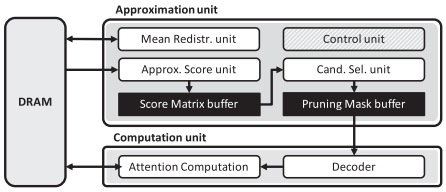

Fig. 6. Overview of HAMMER architecture.

redistribution centers the resulting values of *Score* to zero, making ReLU cut off about half of the non-important values in *Score*.

However, the accuracy still degrades compared to the baseline EXP on a high pruning ratio. To address such limitation, we consider another characteristic of softmax, "winner-takes-all". Because exponential function increases rapidly as the input value increases, the largest value takes over the overall softmax output. To emulate such a property of exponential, we modify the ReLU function as shown in Fig. 5. We increase the slope of ReLU by shifting the output to the left by 1, 2, and 3 as input increases. This modification further improves the approximation quality with minimal hardware overheads (i.e., lightweight shifters for implementation).

Our proposal differentiates itself from previous linear piecewise approximations in that it has a much smaller range than the naive, linear approximation of the exponential function. Specifically, our shifted ReLU requires much less bit-width to store the results while still preserving much more information.

4 HAMMER Architecture

Fig. 6 shows HAMMER's overall architecture and its key pipeline stages, which consist of two parts: approximation and computation. First, the approximation unit computes the approximated *Score*, and with the user-defined threshold, it selects the important *Query*, and *Key* pairs. These pairs are encoded as a pruning mask. The computation unit takes this pruning mask and computes only the pairs marked by the approximation unit.

Mean Computation Unit. Mean computation unit first accumulates the input *Key* vectors sequentially and divides them by the inputs' sequence length to attain the mean value. Then it subtracts this mean from the input *Key* vectors for mean-redistribution. The cost of this operation is amortized over multiple rows in *Query* and incurs negligible overhead both for latency and area (detailed in Section 5).

Approximate Score Unit. Fig. 7 describes the architecture of the approximate score and candidate selection units. The approximate score unit takes the *Query* and *Key* to quantize them and compute the approximated *Score*. First, it loads the input *Query* and *Key* vectors and quantizes them into 2-bit. Then we compute the dot product between them. Because they are quantized into a power of 2 (e.g., -8, -1, 1, and 8), multiplication can be replaced with a few shift operations and reduced with integer adders. Then we apply the proposed shifted ReLU function to the dot product result. The computed values are stored in the score buffer and accumulated in the accumulation buffer for further use in the following pipeline such as the candidate selection unit.

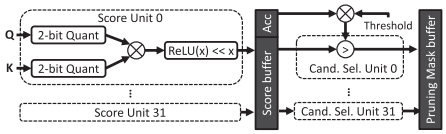

Fig. 7. Microarchitecture of approximate score and candidate selection unit.

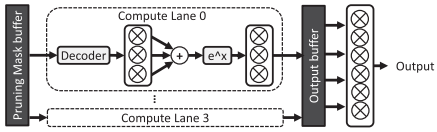

Fig. 8. Microarchitecture of attention computation unit.

Candidate Selection Unit. Once each *Score* is approximately derived, we compute the softmax and compare its value to the user-defined threshold. We avoid the division operation in softmax by comparing each element with the threshold multiplied by the accumulated *Score* value. If it is greater than the threshold, we set the bit in the pruning mask for further computation in the following computation unit.

Attention Computation. Fig. 8 describes the architecture of the attention computation unit. First, the decoder unit scans the pruning mask sequentially to find the marked bit. Then it loads the corresponding input *Query* and *Key* vectors, computes the dot-product between them, and applies an exponential function to its result. We use fixed-point arithmetic units for dot-product and floating-point arithmetic units for exponential. The result is then multiplied with the corresponding input *Value* vector, and accumulated on the output buffer. The accumulated vector on the output buffer is then multiplied with the reciprocal of the sum of the exponential *Score* values for computing the final output.

5 Evaluation

In our experiment, we use BERT-Large[9], ALBERT-Large[10], and RoBERTa-Large[11], which represents NLP models using the transformer architecture. We used the open-source implementation of these models from Hugging Face and fine-tuned these models until we achieved similar accuracies ($\leq \pm 0.5\%$) reported in [2]. We run these models on the datasets SQuAD 1.0 and 2.0 [4], [12], which are representative datasets for question-answering tasks. As a baseline, we use a single NVIDIA Titan Xp equipped with 12GB of GDDR5X and Intel i7-6850k at 3.6GHz equipped with 128GB of DDR4. We also use the state-of-the-art `Self-attention` accelerator ELSA [2] as the baseline. We implement the algorithm of HAMMER with PyTorch to report the algorithmic performance and cycle-level simulator to report the hardware performance. The hardware microarchitecture of HAMMER is implemented using Verilog and synthesized with Synopsys Design Compiler with a commercial 65nm standard cell library to report area and energy. For hardware performance evaluations, to match the performance of the GPU (Titan Xp) with a peak performance of 12 TFLOPS, 12 instances of HAMMER each with 1 TOPS of peak performance are used with batch-level parallelism.

Algorithmic Performance. Fig. 9 shows the accuracy of our HAMMER algorithm compared to ELSA. We sweep the user-defined threshold in the approximation to control the pruning ratio and report the accuracy over the test set. Compared to ELSA, HAMMER achieves a greater pruning ratio at iso-accuracy for all studied benchmarks.

Hardware Performance. Fig. 10a shows HAMMER's performance compared to GPU at 99% of the baseline accuracy and the accuracy loss of 1% and 2.5% each. The HAMMER resulted in a larger speedup than ELSA thanks to its higher pruning ratio at iso-accuracy. The HAMMER outperforms ELSA $1.39\times$ on average with an accuracy drop of 1%, and $1.28\times$ on average with an accuracy drop

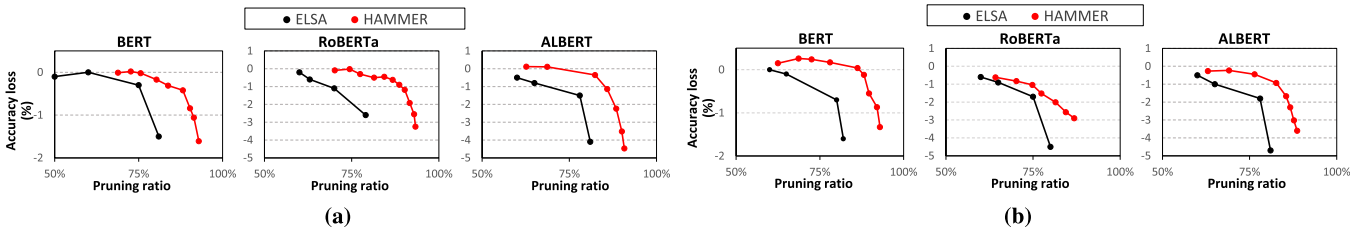

Fig. 9. Accuracy (F1 score) comparison between ELSA and HAMMER evaluated on (a) SQuADv1 and (b) SQuADv2

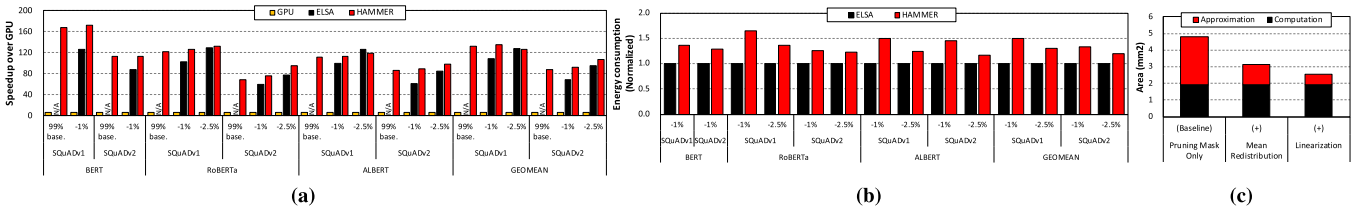

Fig. 10. Hardware performance evaluations (a) Speedup over GPU (b) Energy efficiency (c) Contribution of each proposed method on area reduction

of 2.5%. The overhead of computing mean-redistribution (not shown in the figure) only takes account of the 8-16% in the overall performance. This is because the processed Key values are reused across the multiple input $Query$ vectors, and the cost of computing means and subtracting them from the Key values are amortized.

Energy. Fig. 10b shows the energy-efficiency of HAMMER over ELSA. We compare the energy-efficiency when the accuracy loss is 1% / 2.5%. Because HAMMER reduces energy consumed in the computation unit (which takes up most of the total energy consumed), it results in greater energy-efficiency by up to 1.5x and 1.3x on average.

Area. Table 1 reports the area of HAMMER and ELSA. Because ELSA is synthesized with 40nm library, we scaled the number reported by ELSA to 65nm node for a fair comparison. For ELSA, the approximation unit accounts for 45% of the total area, which is a significant overhead. With HAMMER, we reduce the area of the approximation unit by 67% and the total area by 27%.

Ablation Study. Fig. 10c shows how each proposed method contributes to reducing the area in the approximation module. With quantization, we can remove the multipliers in the approximate score units, reducing the overall area by 35%. With our proposed softmax linearization, we can further reduce the size of on-chip buffers and replace floating-point arithmetic units in the approximation module with fixed-point ones, again reducing the overall area by 19%.

6 Related Work

Recent literature explored various optimization strategies to accelerate `Self-attention` using approximate computing. However, prior work based on input sorting [1] and hashing [2] require large on-chip memory, incurring high area overheads. There exists some prior work employing lightweight approximation schemes but they either require additional training [3] or huge approximation module [13] to compensate for high accuracy loss. HAMMER presents a lightweight approximation module that does not sacrifice algorithmic accuracy, achieving 8.9x, 1.7x, 2.6x, and 4.0x higher performance/area versus [1], [2], [3], and [13], respectively, without requiring additional training.

TABLE 1
HAMMER Area Breakdown

Module	Component	Area (mm^2)	
		ELSA / 65 nm ($estimated$)	HAMMER / 65 nm ($measured$)
Approximation	Mean Comp.	-	0.044
	Candid. Sel.	1.008	0.459
	On-chip buffer	0.472	0.160
	Total	1.480	0.651
Computation	-	1.817	1.907
Total	-	*3.297*	*2.558*

7 Conclusion

`Self-attention` mechanism is the core of various modern NLP neural networks. This paper presents HAMMER, the accelerator for `Self-attention`, which exploits the sparsity in the `Self-attention` mechanism to prune unnecessary computations. HAMMER effectively approximates the existing `Self-attention` mechanism to spot unimportant computations beforehand. With low-bit quantization, mean redistribution, and softmax linearization, our proposed approximation scheme results in minimal overhead in terms of power and area. Compared to the previous state-of-the-art `Self-attention` accelerator, HAMMER achieves a $1.2-1.6\times$ and $1.2-1.5\times$ improvement in performance and energy-efficiency respectively.

References

[1] T. J. Ham et al., "A: Accelerating attention mechanisms in neural networks with approximation," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, 2020, pp. 328–341.
[2] T. J. Ham et al., "ELSA: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks," in *Proc. IEEE Int. Symp. Comput. Architect.*, 2021, pp. 692–705.
[3] H. Wang, Z. Zhang, and S. Han, "SpAtten: Efficient sparse attention architecture with cascade token and head pruning," in *Proc. IEEE Int. Symp. High-Perform. Comput. Architect.*, 2021, pp. 97–110.
[4] P. Rajpurkar et al., "SQuAD: 100,000+ questions for machine comprehension of text," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2016, pp. 2383–2392.
[5] M. Caron et al., "Emerging properties in self-supervised vision transformers," in *Proc. Int. Conf. Comput. Vis.*, 2021, pp. 9630–9640.
[6] M. Guo et al., "LongT5: Efficient text-to-text transformer for long sequences," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics*, 2022, pp. 724–736.
[7] M. Zaheer et al., "Big Bird: Transformers for longer sequences," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 17283–17297.
[8] A. Vaswani et al., "Attention is all you need," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6000–6010.
[9] J. Devlin et al., "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics*, 2019, pp. 4171–4186.
[10] Z. Lan et al., "ALBERT: A lite BERT for self-supervised learning of language representations," 2020, *arXiv:1909.11942*.
[11] Y. Liu et al., "RoBERTa: A robustly optimized bert pretraining approach," 2019. [Online]. Available: https://arxiv.org/abs/1907.11692
[12] P. Rajpurkar et al., "Know what you don't know: Unanswerable questions for SQuAD," in *Proc. Assoc. Comput. Linguistics*, 2018, pp. 784–789.
[13] L. Lu et al., "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," in *Proc. Int. Symp. Microarchitecture*, 2021, pp. 977–991.