Energon: Toward Efficient Acceleration of Transformers Using Dynamic Sparse Attention

Zhe Zhou, Junlin Liu, Zhenyu Gu, and Guangyu Sun, *Member, IEEE*

***Abstract*—In recent years, transformer models have revolutionized natural language processing (NLP) and shown promising performance on computer vision (CV) tasks. Despite their effectiveness, transformers' attention operations are hard to accelerate due to the complicated data movement and quadratic computational complexity, prohibiting the real-time inference on resource-constrained edge-computing platforms. To tackle this challenge, we propose Energon, an algorithm-architecture co-design approach that accelerates various transformers using dynamic sparse attention. With the observation that attention results only depend on a few important query-key pairs, we propose a mix-precision multiround filtering (MP-MRF) algorithm to dynamically identify such pairs at runtime. We adopt low bitwidth in each filtering round and only use high-precision tensors in the attention stage to reduce overall complexity. By this means, we significantly mitigate the computational cost with negligible accuracy loss. To enable such an algorithm with lower latency and better energy efficiency, we also propose an Energon co-processor architecture. Elaborated pipelines and specialized optimizations jointly boost the performance and reduce power consumption. Extensive experiments on both NLP and CV benchmarks demonstrate that Energon achieves 168× and 8.7× geo-mean speedup and up to 10^4× and 10^3× energy reduction over Intel Xeon 5220 CPU and NVIDIA V100 GPU, respectively, Compared to state-of-the-art attention accelerators SpAtten and A^3, Energon also achieves 1.7× and 1.25× speedup, and 1.6× and 1.5× higher energy efficiency.**

***Index Terms*—Computer architecture, deep learning, hardware acceleration, transformers.**

I. Introduction

BENEFITING from the powerful attention mechanism, transformer models, such as Seq2seq [65], BERT [12], GPT-2 [47], XLNet [70], T5 [48], and other variants [28], [29], [31], have achieved leading-edge performance on various natural language processing (NLP) tasks, such as question-answering, text classification, machine translation, etc. Besides, transformers also show promising performance on many computer vision (CV) tasks, including image classification [13], [64], object detection [4], [78], and even video comprehension [54], [61]. In brief, transformers are becoming the de facto substitutes for traditional RNNs and CNNs in a broad range of scenarios.

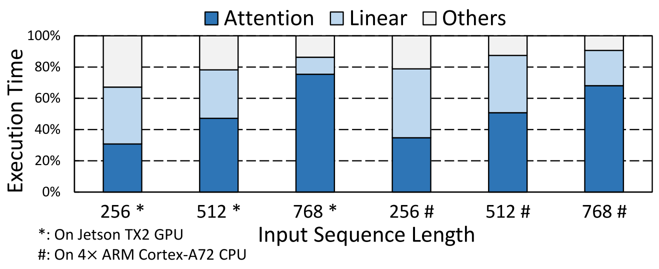


Fig. 1. Execution time breakdown of the BERT-base model on two typical edge-computing platforms.

Despite their effectiveness, it is still challenging to deploy transformers on resource-constrained devices. The greatest difficulty comes from the attention operations, which involve complicated data movement [15], [68] and bear quadratic computational complexity concerning the input sequence length [3], [25], [46], [52]. Therefore, for many tasks having long input sequences, such as question-answering and image-classification, it is always difficult to realize low-latency inference. For instance, we have profiled the execution time of the BERT-base model [12] on two popular edge-computing platforms. As shown in Fig. 1, the attention operations take up about half of the total execution time as the sequence length grows to 512. When the input length reaches 768, attention operations even dominate the total execution (70% on average). Note that the attention overhead is becoming more prominent as weight-pruning and quantization methods have significantly reduced the complexity of linear layers in transformers [55], [71], [72], [74]. Specialized NN accelerators have also been proposed to handle the nonattention parts [6], [16], [71]. Therefore, how to process attention operations efficiently is the key problem of transformers acceleration.

Some recent works propose to co-design attention algorithms and accelerator architectures to mitigate the attention overhead. They mainly exploit the sparsity of attention to speed up the execution. For instance, A^3 [15] leverages several approximation strategies to avoid computing near-zero scores to reduce the computational overhead. SpAtten [68] proposes

Manuscript received 2 August 2021; revised 23 February 2022; accepted 6 April 2022. Date of publication 27 April 2022; date of current version 22 December 2022. This work was supported in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2021B0101310002; in part by NSF of China under Grant 61832020, Grant 62032001, and Grant 92064006; in part by the Beijing Academy of Artificial Intelligence (BAAI); and in part by 111 Project under Grant B18001. This article was recommended by Associate Editor F. Liu. *(Corresponding author: Guangyu Sun.)*

Zhe Zhou, Junlin Liu, and Guangyu Sun are with the Center for Energy-Efficient Computing and Applications, Peking University, Beijing 100871, China (e-mail: gsun@pku.edu.cn).

Zhenyu Gu was with the DAMO Academy, Alibaba Inc., Beijing 100102, China (e-mail: guzhenyu@gmail.com).

Digital Object Identifier 10.1109/TCAD.2022.3170848

a cascaded token pruning mechanism that progressively prunes unimportant tokens to reduce the overall complexity. However, these two solutions still have intrinsic drawbacks. Specifically, A^3 needs to load all the data on-chip to perform approximate computation, failing to reduce off-chip DRAM access. The cascaded token pruning used in SpAtten successfully reduces both computation and DRAM access, but it is a coarse-grained strategy that does not support dynamic pruning for different attention heads. Our experiments reveal that token pruning can hardly achieve acceptable accuracy without model retraining. Moreover, previous works only conduct evaluations on NLP tasks. Their performance on vision tasks, e.g., image classification, remains to be explored.

To overcome these problems, we propose Energon,[1] an algorithm-architecture co-design solution that efficiently accelerates transformers using dynamic sparse attention. With the observation that attention results mainly depend on a few important query-key pairs, we propose a novel mix-precision multiround filtering (MP-MRF) algorithm to identify these pairs at runtime. We adopt low-bitwidth operations in each filtering round. Only the finally selected pairs are used for sparse attention with high-precision tensors. By this means, we reduce 4× to 8× computation with negligible accuracy loss. To enable such an algorithm with lower latency and better energy efficiency, we also design a co-processor architecture named Energon. It boosts the performance and saves DRAM access through elaborated pipelines and several optimization strategies. In addition, we propose a performance model to analyze the bottlenecks of Energon architecture under different situations so as to guide the hardware configuration. Extensive experiments on both NLP and CV benchmarks show that, on average, Energon gets 37× and 440× geo-mean speedup and achieves 340× and 2951× geo-mean energy reduction compared to TX2 GPU and ARM-A72 CPU, respectively. Besides, to meet the requirements in cloud-computing scenarios, we also present an Energon-server, which shows 168×, 8.7× geo-mean speedup, and 10^4×,10^3× energy-efficiency compared to Intel Xeon CPU and NVIDIA-V100 GPU. We further compare our design against the state-of-the-art attention accelerators and prove that Energon achieves on average 1.7×, 1.25× speedup and 1.6×, 1.5× energy efficiency over SpAtten and A^3. To summarize, we have the following main contributions.

1) We propose an MP-MRF algorithm to realize dynamic sparse attention in transformers (Section III). Without retraining, it achieves 4× to 8× pruning ratio with negligible accuracy loss on both NLP and CV tasks (Section V-A).
2) We design an Energon architecture to accelerate the proposed algorithm. It functions as a co-processor that is plug-in compatible with other NN accelerators. We carefully design the pipelines and propose several architectural optimizations to boost the performance and reduce energy consumption. A performance model is also introduced to help determine the hardware configurations (Section IV).
3) We prove that Energon achieves multiple orders of magnitude speedup and energy reduction over commodity CPUs/GPUs. Energon also outperforms state-of-the-art attention accelerators SpAtten and A^3 (Section V).

[1]In the famous film series *Transformers*, Energon is the preferred fuel of the Transformer race.

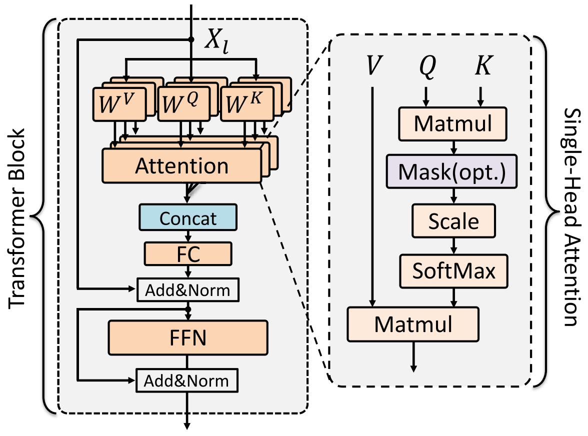


Fig. 2. Standard transformer block. W^Q, W^K, W^V are projection weights to project the input feature X_l to Query (Q), Key (K), and Value (V) features.

II. Background and Motivation

A. Transformer Algorithms

Transformers, first proposed by Google [65], have demonstrated leading-edge performance on a variety of NLP tasks, including discriminative language understanding tasks [12], [28], [31] and generative language modeling tasks [47], [48], [65]. Surprisingly, transformers are also gaining more and more popularity among CV tasks [4], [13], [54], [60], [78]. Thanks to their outstanding effectiveness and generality, transformers are considered substitutes for traditional RNNs and CNNs in many real-world scenarios.

Unlike traditional DNNs [27], [34], [56], transformer models are generally built by stacking several transformer blocks. As illustrated in Fig. 2, a standard transformer block consists of three key components, namely, linear layers [including projection layers, fully connected (FC) layers, and the feed-forward network (FFN)], multihead attention (attention), and normalization layers (Norm). For the lth block of a transformer model, the input is a sequence of n vectors (tokens), denoted as $X_l \in \mathbb{R}^{n\times d_{\text{in}}}$ where n and d_{in} are the sequence length and the input feature dimension. Three linear projection weights $\{W^Q, W^K, W^V\} \in \mathbb{R}^{d_{\text{in}}\times d}$ project the input X_l to query, key, and value, denoted as $\{Q, K, V\} \in \mathbb{R}^{n\times d}$, where d is the hidden feature size. Attention is then performed on these features to capture long-term dependencies of the input sequence.

As described in Algorithm 1, an attention layer has in total H different attention heads, each processing a chunk of $Q, K,$ and V. Therefore, we first split $Q, K,$ and V to H chunks, respectively. Single-head attention is then applied to each chunk. Just as illustrated in the right part of Fig. 2, the hth head computes the outputs as follows:

$$\text{Attention}(Q_h, K_h, V_h) = \text{Softmax}\left(\frac{Q_h \cdot K_h^{\mathsf{T}}}{\sqrt{d_h}}\right)V_h. \quad (1)$$

The *attention score*, namely, the scaled dot-production result of query Q_h and key K_h, is passed to a *Softmax* function to get the *attention probability* (we use d_h to denote the feature size of one head chunk). Then, the probability matrix is multiplied

Algorithm 1: Multihead-Attention

```
1  Input: $\{Q, K, V\} \in \mathbb{R}^{n\times d}$, number of heads $H$;
2  Split $Q, K, V$ to $H$ chunks respectively;
3  $d_h = d/H$;
4  for head_index $h \leftarrow 0$ until $H$ do
5      attention_score $\leftarrow Q_h \cdot K_h^T / \sqrt{d_h}$;
6      attention_score $\in \mathbb{R}^{n\times n}$;
7      for row_index $i \leftarrow 0$ until $n$ do
8          attention_prob[$i$] $\leftarrow$ Softmax(attention_score[$i$]);
9      end
10     attention_result[$h$] $\leftarrow$ attention_prob$\cdot V_h$;
11 end
12 output $\leftarrow$ concat(attention_result);
13 Return: output $\in \mathbb{R}^{n\times d}$;
```

with value tensor V_h to derive the single-head attention result, which has a shape of $n \times d_h$. Each head computes *attention result* individually. The results are concatenated to form the final results. Note that in the remainder of this article, we also use Q, K, and V to denote the single-head inputs Q_h, K_h, and V_h and use d to represent a single head's feature dimension for simplicity. Apart from multihead-attention, the remaining operations, such as Normalization and FFN, have been widely applied in traditional DNNs. To be specific, the FFN has two FC layers

$$\mathrm{FFN}(X_l) = \mathrm{GELU}(X_l W_1 + b_1) W_2 + b_2 \tag{2}$$

where W and b represent the weight and bias, and GELU is a special activation function [18].

Since the attention layer conducts pairwise dot-productions among n vectors, the complexity should be $O(n^2 d)$, while linear layers' complexity is about $O(nd^2)$, because they process each input token individually. Given that d is usually a fixed value, the overhead of attention operations grows more rapidly than the linear parts as the length of input sequence n increases. Our profiling results in Fig. 1 have vividly shown this trend. On both embedded GPU and CPU platforms, the attention operations take up more than half of the total execution time as n increases larger than 512 and dominate the execution time when n is set to 768. That is to say, for tasks requiring long input sequences, the attention operations will become the bottleneck. Moreover, the challenge of attention acceleration is becoming increasingly prominent as weight pruning [17], quantization [71], [72], and specialized NN accelerators [6], [7], [16], [71] can provide efficient acceleration of linear layers in transformers. Therefore, it is urgent to devise solutions to execute attention operations more efficiently.

B. Sparse Attention

Several algorithms and accelerators focused on exploring the sparsity in attention layers have been proposed to mitigate the attention overhead of transformers. As described in Algorithm 1, the dot production of Q and K generates the $n \times n$ attention score matrix. Since a *Softmax* function is applied to each row of the score matrix to derive the attention probability, the chances are that only a small fraction of scores in each row will produce a large probability, and most of the dot-production scores are too small to affect the final attention results.

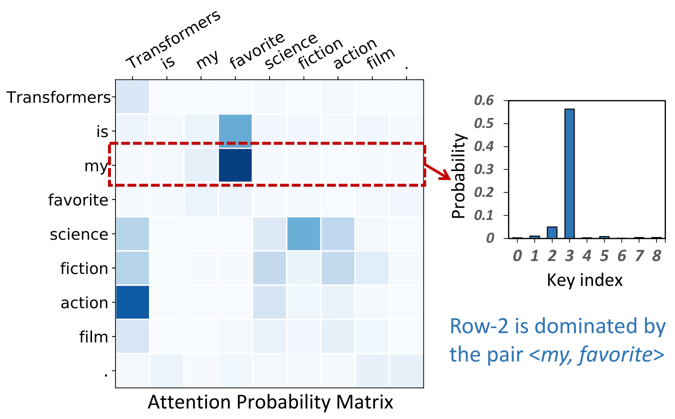


Fig. 3. Sparsity in attention probability matrix. In the annotated row, the data distribution is dominated by the query-key pair <*my, favorite*>.

Fig. 3 illustrates one of the probability matrices extracted from the BERT-base model trained on the SST-2 dataset [58]. For most of the rows, only a small portion of elements have a large probability. For instance, in the annotated row, the probability of pair <*my, favorite*> dominates the distribution. If we only use those "important" query-key pairs for attention and ignore the others, we can save much computation. Moreover, we can only load the selected keys and values on-chip to reduce total memory access. Therefore, the question becomes: *How to find the important query-key pairs?* Previous works' answers to this problem have formed two trends: 1) content-independent sparsity and 2) content-based sparsity.

Content-Independent Sparsity: A lot of prior works simply force the attention matrix to be sparse through applying fixed, predefined patterns such as local windows [9], strided patterns [3], [9], blockwise patterns [46], or combining two or more distinct patterns [9], [19]. However, these heuristic methods fail to find the important query-key pairs explicitly since they do not take the input features' content into consideration. Therefore, the performance is always limited [63].

Content-Based Sparsity: Unlike content-independent methods, content-based methods dynamically determine the sparsity pattern through analyzing the inputs. To name a few, Reformer [25] uses local-sensitive-hashing (LSH) to select the query-key pairs that are close in the hashing space. It reduces the attention complexity to $O(n \log n)$. Meanwhile, Routing-transformer [52] leverages clustering algorithms to find the important pairs before attention. It reduces the computational complexity from $O(n^2 d)$ to $O(n^{1.5} d)$. However, both methods have intrinsic drawbacks: the LSH hashing needs to be performed multiple rounds (up to 8 according to the experiments) to guarantee accuracy, and the clustering algorithm itself is too complex for embedded accelerators. Some other works [62], [75] also propose dynamic sparse attention methods. However, they only focus on algorithm improvements and seldom consider hardware designs.

C. SW/HW Co-Design Approaches

We notice that a few previous works attempt to accelerate attention operations through software-hardware co-design. As mentioned before, A^3 [15] accelerates attention operations in neural networks with algorithmic approximation and hardware

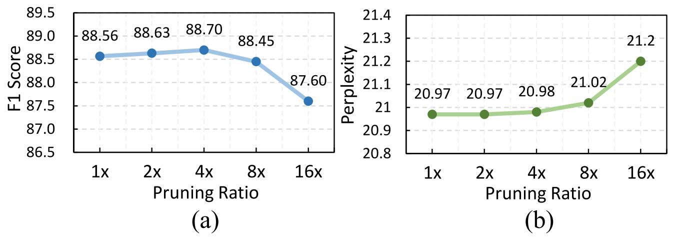


Fig. 4. Accuracy versus pruning ratio using Top-k pruning. (a) Question-answering task on Squad-v1 dataset. (b) Language-modeling task on Wikitext.

specialization. However, A^3 only saves computation and cannot reduce DRAM access. Another recent work SpAtten [68] proposes a cascaded token pruning mechanism to prune unimportant tokens layer by layer, which reduces both computation and DRAM access. However, token pruning does not support fine-grained dynamic pruning for different attention heads. According to our evaluation, it fails to achieve acceptable accuracy in the absence of retraining. Moreover, these two works are only evaluated on NLP tasks. Whether sparse attention works on CV tasks remains to be discussed.

III. Energon Algorithm

We argue that an ideal sparse attention mechanism should meet three following goals: 1) it should achieve a high pruning ratio and maintain the models' accuracy without retraining, as the training dataset is always inaccessible in deployment scenarios; 2) it should be compatible with various transformer models, including NLP and CV models; and 3) it should be hardware friendly and have the potential to reduce both computation and DRAM access. With these goals, we propose a dynamic sparse attention algorithm using a novel multiround filtering strategy. Details are introduced as follows.

A. Top-k Pruning: Baseline

We begin our discussion with a question: *How to identify the query-key pairs that will produce large attention probabilities at runtime?* Given an attention score matrix, a straightforward way is to keep the k-largest elements in each row to calculate the probability. We view this method as the baseline solution. To explore the efficiency of top-k pruning, we evaluate it on two representative tasks: 1) question-answering task on the SQuAD-v1 dataset [49] using the BERT-base model [12] and 2) language-modeling task on the Wikitext-2 dataset [32] using the GPT-2 model [47]. The evaluation code and pre-trained models are provided by the popular Hugging-Face Transformers [20] project. For each row of the attention score matrices, we sort and select the k-largest scores to conduct *Softmax*. The unselected elements are set to zero. We follow SpAtten's [68] practice and do not prune the first two blocks, since the first few blocks have not captured high-level relations between tokens and therefore, do not show obvious sparse patterns. Without any retraining, the accuracy [F1-score for task 1), higher is better; Perplexity for task 2), lower is better] under different pruning ratios are plotted in Fig. 4. As we can see, when 87.5% of the total keys of each row are pruned (8$\times$ pruning ratio), the two tasks only bear negligible accuracy loss (-0.12% for SQuAD and $+0.05$ for Wikitext). Even with a 16$\times$ pruning ratio, the accuracy loss is within 1%.

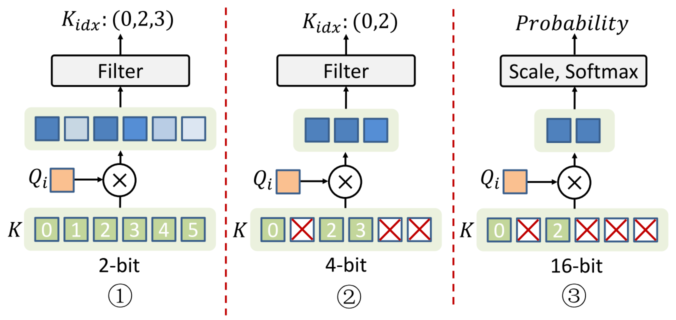


Fig. 5. Illustration of MP-MRF strategy. Round ①: filter keys using 2-bit tensors. Round ②: filter the selected keys further using 4-bit tensors. Round ③: use the remaining keys to conduct high-precision sparse attention.

Such results demonstrate the huge potential of leveraging the sparsity of attention matrices to reduce the attention overhead.

However, the naïve top-k pruning has two obvious drawbacks. First, the pruning is conducted after the calculation of attention scores, which only saves *Softmax* and *attention_prob* $\times$ *V* computation. We still have to conduct the $O(n^2)$ scaled dot production, namely, $Q \cdot K^T/\sqrt{d}$. Second, top-k pruning requires specialized sorting engines to select the k-largest elements. For example, both A^3 and SpAtten spend much effort designing the top-k engines. We argue that a simpler method is preferred to help select the important query-key pairs.

B. Mix-Precision Multiround Filtering

To overcome the aforementioned challenges, we propose a novel MP-MRF method. As Fig. 5 shows, for each query Q_i, we search for the important keys by setting multiple filtering rounds. In each round, we calculate the dot-production scores using low-precision tensors and select the keys whose scores are greater than a threshold. Only the finally selected keys are used for sparse attention with high precision. We introduce the details of MP-MRF mainly from the four following aspects.

1) *Reduce Dot-Production Overhead:* As pointed out above, the baseline top-k pruning still has to compute the whole *attention_score*. To mitigate the computational overhead, we can compute the dot-production $Q \cdot K^T$ with low-bit arithmetic. However, directly applying low-bit quantization (e.g., ternary quantization) to the Q and K tensors will significantly degrade the model accuracy because the attention features are sensitive to quantization errors [71], [74]. To achieve a balance between efficiency and accuracy, we design a mix-precision multiround filtering (MP-MRF) strategy. Instead of the one-round top-k pruning using high-precision tensors, we set multiple filtering rounds. In the beginning, we adopt extremely low bitwidth to compute $Q \cdot K^T$ and select the pairs with large scores. Although low-bit filtering is not that accurate, it still has good coverage if the pruning ratio is conservative (e.g., 50%). In the next round, we perform incremental filtering based on the selected pairs of the previous round. Since the remaining pairs tend to have closer scores, we use higher precision (more bits) to recompute the remaining pairs' dot production. We do such filtering iteratively. After the last round, the finally selected keys in each row are used to perform high-precision sparse attention.

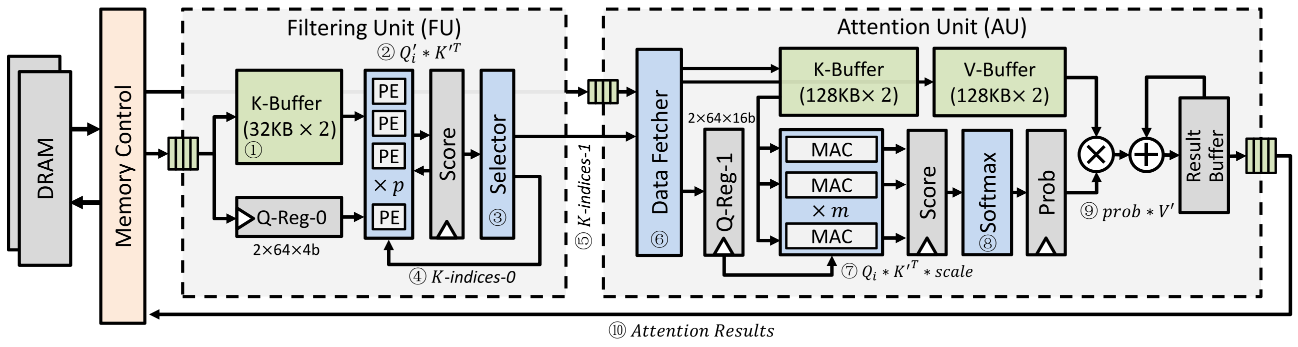


Fig. 6. Energon architecture overview. The FU performs multiround mix-precision filtering to select important keys for each query. The indices of selected keys are fed into the AU for high-precision sparse attention. We adopt ping-pong buffers in both units to hide the memory access overhead as much as possible.

2) *Avoid Top-k Selection:* Another problem is how to avoid the top-k selection. Instead of sorting all the scores, we can use mean filtering [59] to search for the important scores. Specifically, in each round, we estimate each row's mean value and only select the query-key pairs whose scores are greater than the mean value. In this way, we can roughly prune about 50% of the elements in each round. Compared to top-k pruning, such a method is much simpler for implementation.

3) *Adjustable Pruning Ratio:* The mean filtering does not allow us to adjust the pruning ratio. However, in some scenarios, we expect high accuracy. In some other scenarios, we may slightly sacrifice accuracy for a higher speed. How to flexibly balance accuracy and pruning ratio according to different requirements is also an important problem. Instead of directly adopting the mean values as thresholds, for round r, we set the threshold of the ith row as θ_i^r

$$\theta_i^r = \begin{cases} \alpha_r \times \max(S_i^r) + (1-\alpha_r) \times \text{mean}(S_i^r), & 0 \le \alpha_r < 1 \\ -\alpha_r \times \min(S_i^r) + (1+\alpha_r) \times \text{mean}(S_i^r), & -1 < \alpha_r < 0 \end{cases} \quad (3)$$

where S_i^r denotes the attention scores of the ith row (the scores already pruned are ignored). We use a parameter α_r to control the threshold, whose value range is $(-1, 1)$. By adjusting α_r, the threshold transits from $\min(S_i^r)$ to $\max(S_i^r)$. Thus, we can control the pruning ratio in each round from 0% to 100%. We describe the overall MP-MRF mechanism in Algorithm 2.

4) *Mix-Precision Quantization:* The proposed MP-MRF requires mix-precision quantization. To make it more efficient, we first perform INT16 quantization and then obtain the low-precision tensors (e.g., INT2 and INT4 tensors) directly by truncating the most significant bits (MSBs) of the INT16 data. We only need to perform the quantization once and fetch a different number of bits to compute each filtering round. Moreover, such a way also enables us to reuse the results among different rounds, which will be discussed later.

We illustrate a two-round MP-MRF in Fig. 5. Through experiments in Section V-D, we observe that two-round filtering with 2-bit tensors in the first round and 4-bit tensors in the second round is both accurate and cost effective for our tasks. In the following discussions, we view the two-round filtering as our default configuration.

Algorithm 2: MP-MRF

```
1  **Input:** Total rounds $R$;
2  Filtering parameters $\{\alpha_0, \dots, \alpha_{R-1}\}$;
3  Bit-width of each round $\{l_0, \dots, l_{R-1}\}$;
4  Quantized $Q, K, V \in \mathbb{R}^{n\times d}$;
5  **for** $i \leftarrow 0$ *until* $n$ **do**
6      $K_{idx} \leftarrow \{0, 1, \dots, n-1\}$;
7      **for** $r \leftarrow 0$ *until* $R$ **do**
8          $Q_i' \leftarrow Q[i]$, load the first $l_r$ bits;
9          $K' \leftarrow$ gather($K_{idx}, K$), load the first $l_r$ bits;
10         $S_i^r \leftarrow Q_i' \cdot K'^T$;
11         Estimate $\theta_i^r$ using Equation 3;
12         $K_{idx} \leftarrow \{j \mid S_i^r[j] > \theta_i^r\}$;
13     **end**
14     $Q_i \leftarrow Q[i]$, load full bits;
15     $K' \leftarrow$ gather($K_{idx}, K$), load full bits;
16     $V' \leftarrow$ gather($K_{idx}, V$);
17     $attention\_prob \leftarrow \text{Softmax}(Q_i \cdot K'^T/\sqrt{d})$
18     $Result[i] \leftarrow attention\_prob \cdot V'$
19 **end**
20 **Return:** $Result$
```

IV. Energon Accelerator

Current commodity CPUs/GPUs and attention accelerators cannot support the proposed dynamic sparse attention mechanism due to the mix-precision computation and specialized pipeline. Therefore, we propose a novel Energon accelerator. It functions as a co-processor that is plug-in compatible with many other NN accelerators.

Our Energon architecture mainly meets two design goals: 1) it supports the proposed MP-MRF with low latency and high throughput. We achieve this goal by designing efficient hardware pipelines and exploring data-level parallelism and 2) it is both energy efficient and area efficient. Since Energon is a co-processor working together with other NN accelerators, it should save on-chip areas. We use specialized inner product unit (IPU) with result-reusable processing elements (PEs) to reduce arithmetic units' overhead to meet such goals.

A. Architecture Overview

Design: Fig. 6 illustrates an overview of the Energon architecture. In general, Energon is composed of two functional units. A filtering unit (FU) is designed for supporting the MP-MRF mechanism (see Section III) and outputs the indices of the important keys for each query. The attention unit (AU) enables high-precision sparse attention, given the K-indices (denotes the indices of selected keys). Energon is equipped with external DRAM, which stores the Q, K, and V tensors and the computed attention results.

Workflow: When Q,K, and V tensors are produced by other processors, they are also quantized and stored in DRAM, with INT4 and INT16 data stored separately. Energon accelerator then computes one attention head at a time and processes each head's queries in a pipelined manner. To be specific, when a head's processing starts, the K and V tensors required by FU (INT4) and AU (INT16) are loaded from DRAM to on-chip buffers, after which the queries are fed into the pipeline. Given a query Q_i, FU filters the important keys through cost-efficient low-precision operations and generates the indices of selected keys for AU. AU performs high-precision sparse attention according to the received K-indices. The attention result of each query will be written back to DRAM immediately. Thus, co-located DNN accelerators can use them to conduct the remaining computation. We introduce more details of each module and the proposed optimizations in the following sections.

B. Filtering Unit

The whole pipeline starts with the FU. As aforementioned, FU is designed to support the multiround filtering algorithm. According to Algorithm 2, in the rth round, FU computes the approximated dot-production scores $S_i^r = Q_i' \cdot K'^T, i \in \{0, \dots, n-1\}$ using low-bit tensors (the adopted bitwidth in round r is denoted as l_r). It then searches for the indices of important keys by comparing them with a dynamic threshold θ_i^r estimated with *min*, *max*, and *mean* values (3). Under our settings, FU needs to perform two filtering rounds in total. The first round is coarse-grained filtering using INT2 tensors. The selected keys are used for fine-grained filtering with INT4 tensors in the second round. Such a mix-precision workflow demands both INT2 and INT4 arithmetic units. To enhance the processing efficiency and save on-chip resources, we propose two specialized designs and optimizations: 1) result-reusable mix-precision IPU and 2) optimized key data layout. The details are introduced as follows.

IPU Design: The IPU marked as ② in Fig. 6 is responsible for computing the inner product $Q_i' \cdot K'^T$ where Q_i' is $1 \times d$ vector and K'^T is $d \times n$ tensor (line 10 in Algorithm 2). As we can see, IPU is composed of multiple PEs working in parallel. Each PE computes d multiplications each time and adds up the results to get an inner production result. To support both INT2 and INT4 processing, we design a novel PE architecture illustrated in Fig. 7. It not only enables mix-precision processing but also reduces the computational complexity through reusing the results. Specifically, supposing $d = 64$ and a PE equips 64 multipliers. Each multiplier is responsible for a 4-bit $\times$ 2-bit multiplication. Instead of using 2-bit Q in the first round (round-0) and 4-bit Q in the second round (round-1), we adopt 4-bit Q in both rounds while still keeping the K to be 2-bit in the first round. Though such a change increases the computational overhead in round-0, it enables us to use the same hardware in both rounds and reuse the round-0 results.

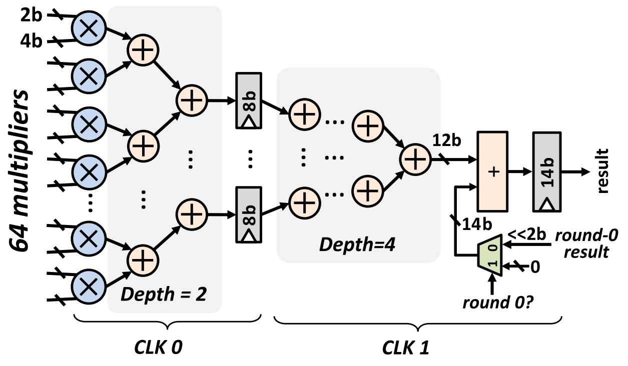


Fig. 7. Architecture of the proposed result-reusable multiprecision PE.

To be specific, given two 4-bit vectors Q_i and K_j^T, we have the following formulation:

$$Q_i * K_j^T = \left(Q_i * K_j[3:2] << 2\right) + Q_i * K_j^T[1:0]$$

where $K_j^T[3:2]$ means truncating the two MSB of each element of K_j while $K_j^T[1:0]$ represents the truncation of the two least-significant bits (LSBs). Recall that we directly use the MSBs of INT4 data as the INT2 quantization. Therefore, if the INT4 operands (Q_i) are the same in both rounds, we can first calculate $Q_i * K_j^T[3:2]$ in round-0, which serves as the round-0 score. Then, in round-1, we calculate $Q_i * K_j^T[1:0]$, which is added with the left-shifted round-0 score to produce the round-1 score. By this means, we can save about half of the computation in round-1. As shown in Fig. 7, we implement such a mechanism by equipping an adder and a shifter to deal with the shift-and-add operation and use a multiplexer to control the data path. In round-0, both 2-bit and 4-bit operands are treated as signed integers, and the output of the adder tree will be added with zero. The results are buffered in a score register (see Fig. 6) and used to select important keys. In round-1, the 2-bit operands are viewed as unsigned integers. The corresponding round-0 scores are loaded, left-shifted, and added to the adder-tree results to obtain the final scores.

As shown in Fig. 7, we insert registers into the adder tree to divide the whole PE into two stages. The multipliers and 2-level adder-tree belong to the first stage, while the 4-level adder tree and the shift-adder belong to the second stage. This design ensures that the latency of each stage is within 1 ns under the evaluation using the FreePDK 45-nm standard library [35]. Therefore, each PE outputs a result every two cycles.

Data Layout: We store the 4-bit keys in a 32 kB$\times$2 (double buffering) K-Buffer. Supposing that there are up to 1024 tokens ($n = 1024$) and $d = 64$ for each head, we store the MSBs and LSBs of each element separately. Each row of the SRAM has 512 bits containing 2-bit features from four tokens, which are transferred at a cycle. IPU reads MSBs rows at round-0 and reads LSBs rows at round-1. To provide enough bandwidth in both rounds, the MSBs and LSBs rows are interleaved to eight single-ported SRAM banks. Moreover, the queries are stored in a register Q-Reg-0, the size of which is set to $2 \times 64 \times 4$ bits = 64B. "2" is for double buffering.

Selector Module: As Fig. 8 illustrates, the selector is responsible for selecting the important K-indices based on the

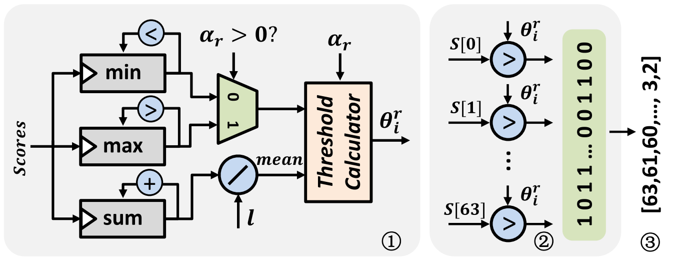


Fig. 8. Architecture of the Selector module. ①: the *min*, *max*, and *sum* of scores are dynamically calculated to estimate the threshold θ_i^r. ②: scores are compared with the threshold in parallel to generate the *K-indices* ③.

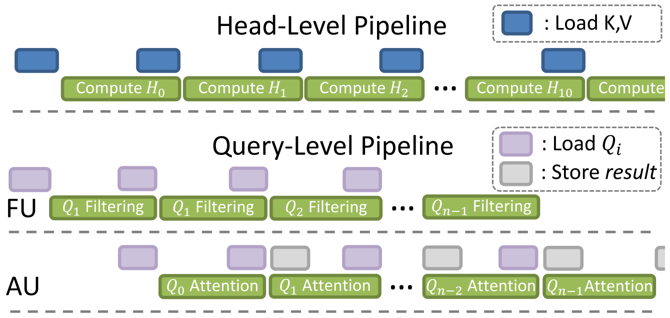


Fig. 9. Two-Level Pipelines. Double QKV buffers enable the overlapping of memory access and computation.

dot-production scores computed by IPU. It first estimates the dynamic threshold θ_i^r according to (3), then compares the scores with the threshold. Finally, the indices of keys whose corresponding scores are greater than θ_i^r are collected as the outputs. Note that Selector calculates *min*, *max*, and *sum* of the scores immediately after IPU outputs any results into the score registers. Thus, it can finish estimating these three values right after all the dot-production scores of a query are calculated. The *sum* value needs to be divided by l (denotes the number of input scores) to get the *mean* value. Then, it derives θ_i^r using a *Threshold Calculator* logic, which conducts the calculation in (3) with the three estimated values and a filtering parameter α_r. After obtaining θ_i^r, the scores are compared with it in parallel (the parallelism is 64 in the figure). The output binary signals are converted to *K-indices*. As Fig. 6 shows, the *K-indices-0* (④) in the first round are fed to IPU to conduct round-1 filtering, while the *K-indices-1* (⑤) are directly sent to AU for high-precision sparse attention.

C. Attention Unit

As shown in Fig. 6, the AU relies on a Data-Fetcher to fetch the required keys and values from external DRAM according to the K-indices provided by FU, and stores them into the K-Buffer and V-Buffer, respectively. Similar to FU, AU also equips a 2KB double-buffered Q-Reg to hold the 16-bit queries. A MAC Array with m parallel MAC units calculates high-precision inner productions. A Softmax module calculates the attention probabilities, whose exponential function is approximated using Taylor expansion [38]. The details about the whole attention pipeline and other optimizations are introduced as follows.

Attention Pipeline: The attention operations are processed in a pipelined manner. At each step, the query Q_i and selected keys K' are read from Q-Reg-1 and K-Buffer individually and sent to a MAC array to perform 16-bit scaled dot production ($Q_i \cdot K'^T \cdot scale$, where scale $= [1/\sqrt{d}]$). The results are buffered in the score registers. A Softmax module (⑧) then processes the attention scores to get the attention probabilities. Each Softmax module consists of eight exponential units working in parallel to improve throughput. After getting probabilities (Prob), AU finally computes attention results by multiplying Prob and the selected values V' through another multiply-accumulate module (⑨). The attention results (⑩) are written back to DRAM. Since we adopt double buffers to overlap the loading and computation of queries, the whole workflow is fully pipelined when processing an attention head.

Unlike SpAtten, we do not double K and V buffers since the loading of K and V and computation are always imbalanced for our long-input cases. The details about a performance model will be discussed in Section IV-D.

On-Demand Fetching: Recall that A^3 needs to load all keys and values on-chip for approximated attention, which involves tremendous DRAM access. However, we notice that after multiround filtering, if the pruning ratio is relatively high, more than half of the keys in each head are "always unimportant" that do not need to be loaded. For instance, when applying 16× pruning, BERT-base on Squad-v1 dataset only needs averagely 47% of keys for attention in each head. Thus, instead of always fetching all the K and V features, we propose an on-demand fetching (ODF) strategy, which only fetches the keys and values needed by each query incrementally.

For each head, we first load the keys and values required by the first query according to the K-indices provided by FU. After one query is processed, the data fetcher module analyzes the required K-indices of the following query and checks whether any keys and values are not in the buffer. If so, only the missed keys and values will be loaded. Moreover, the keys already in K-Buffer will be computed immediately to overlap DRAM access with computation for each query.

D. Performance Model

According to Energon's workflow, there are two processing pipelines: 1) the head-level pipeline and 2) the query-level pipeline. As illustrated in Fig. 9, each head loads the required K and V tensors on-chip and then starts computing. After finishing one head, the next head will load the K and V and continue the computation. We call it head-level pipeline. Within each head, the queries' loading and the computation are overlapped for both FU and AU, since double Q-Regs are adopted. The filtering and attention for each query are also pipelined between FU and AU. We call such a workflow query-level pipeline.

For different workloads, the sequence lengths, embedding dimensions, and pruning ratios vary a lot. Then, what will be the bottleneck of the pipeline? We propose a performance model to answer this question. Assuming that the DRAM bandwidth is B bytes per cycle. Loading K and V data of each head costs about $t\_load = [(4.5 \times d \times n)/B]$ cycles, where 4.5 is derived by loading 4 bytes for both K and V elements in AU and 0.5 byte for each K element in FU. Supposing the pruning ratio is β, the fully pipelined AU processes each

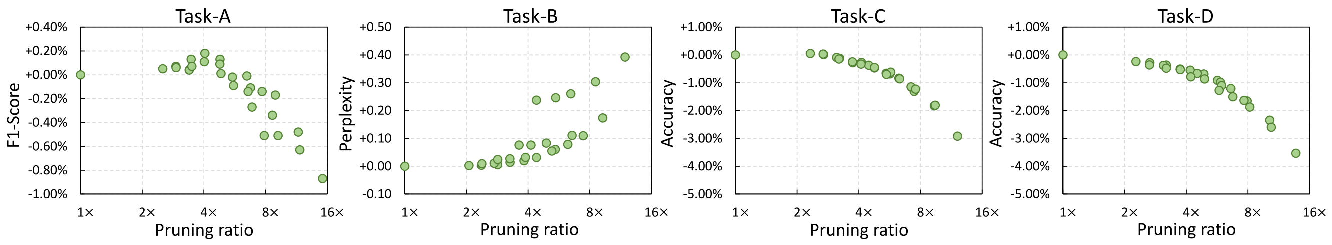


Fig. 10. Accuracy versus pruning ratio exploration using the proposed MP-MRF algorithm.

query using $\lceil(2 \times \beta \times n)/m\rceil$ cycles as the MAC array outputs m results every two cycles. Thus, the computation of each head costs $t\_comp = \lceil(2 \times \beta \times n \times l)/m\rceil$ cycles, where l is the query length. For text-generation tasks with cached K and V (enable `use_cache` in the hugging-face implementation), $l = 1$, otherwise, $l = n$. Thus, the loading-to-computation ratio is roughly $(t\_load/t\_comp) = \lceil(2.25 \times d \times m)/(B \times \beta \times l)\rceil$. For some typical values, say $d = 64$, $m = 8$, $l = 512$, and $\beta = 0.25$, if the HBM memory with 512 GB/s bandwidth (similar to the setup in SpAtten) is equipped and the functional units runs at 1 GHz, then the loading-to-computation ratio is merely 0.017. If low-bandwidth DRAM like LP-DDR3 is equipped, we assume the bandwidth is 25.6 GB/s (two channels), then the ratio is just 0.35. That is to say, for long input sequences, the loading time is usually much less than the computation time. Thus, the head-level pipeline is computation bounded. Double buffers can hardly bring too much performance gain. We can therefore disable half of the double K-V buffers through clock gating to save energy. For middle and short input sequences, say $l = 128$, the $t\_load : t\_compute$ ratio becomes 1.44 (with LP-DDR3 memory). Considering that the data loading and computation cycles are of the same order of magnitude, we still enable double buffering to maximum throughput. Therefore, we can always use the $t\_load : t\_compute$ ratio to decide whether to enable double buffering under different configurations and workloads to either save energy or optimize throughput.

To balance the query-level pipeline, the filtering cycles should match the attention cycles for each query. Since the IPU parallelism is p, it computes each Q_i in $t\_comp' = \lceil(2 \times (1+\gamma) \times n)/p\rceil$ cycles, where γ denotes the pruning ratio in round-0. Considering that the Selector's cycles are negligible if it is highly parallelized, we directly use t_comp' to estimate the filtering cycles for each query. We let t_comp' = t_comp for balancing the FU-AU pipeline, then we derive $(m/p) = \lceil\beta/(1+\gamma)\rceil$. Since $\lceil\beta/(1+\gamma)\rceil$ is always much smaller than 1 according to our experiments, FU should have higher parallelism than AU to balance the whole pipeline. Such a goal is easy to achieve since FU only involves cost-efficient low-precision arithmetic units, which are easy to scale up. We leverage such a performance model to determine hardware configurations in Section V-B.

V. Evaluation

A. Algorithm Evaluation

Benchmarks: To demonstrate the efficiency of the proposed dynamic sparse attention algorithm, we evaluate it on both NLP and CV tasks. We prepare four representative benchmarks listed in Table I. The aforementioned NLP tasks, namely, the question-answering task on SQuAD-v1 [49] dataset and the language-modeling task on Wikitext-2 [32] dataset, serve as tasks A and B, respectively. We also adopt two image-classification datasets: 1) CIFAR-100 [26] and 2) ImageNet [11]. The input sequence lengths are also listed in the table. SQuAD has variable sequence lengths among samples, and the 95th percentile of sequence length is 304 (average length is 176). Task-B with Wikitext-2 dataset has 1024 tokens, and the two CV datasets all have 577 input tokens. We do not adopt GLUE dataset [66] for evaluation, because each GLUE benchmark task only has a limited sequence length ($n < 100$). Obviously, for such tasks, the attention operations will not be the bottleneck.

TABLE I
Benchmarks

Task	Type	Dataset	Model	Sequence Length
A	NLP	SQuAD-v1 [49]	BERT-base [12]	304 (95th pctl.)
B	NLP	Wikitext-2 [32]	GPT-2 [47]	1024
C	CV	CIFAR-100 [26]	ViT-B/16 [13]	577
D	CV	ImageNet [11]	ViT-L/16 [13]	577

Model Setup: We use BERT-base for Task-A, GPT-2 for Task-B, and adopt ViT-B/16 and ViT-L/16 [13] for Task-C and Task-D. All the models except for ViT-L/16 have 12 transformer blocks, each having 12 attention heads. The ViT-L/16 model has 24 blocks with 16 heads in each block. For all the models, each head's feature dimension d is 64. We implement the two NLP models using the hugging-face transformers project, while implement the ViT models using the Pytorch-ViT project [22]. The input image resolution of Task-C and Task-D are resized to 384×384. They are chunked into 576 patches and prefixed with a `Start` token to serve as input sequence. All the models are trained on the officially released pretrain weights with default training parameters. We make sure that the trained models have the same accuracy as the claimed performance in these projects. We then modify the code to support the proposed dynamic sparse attention.

Accuracy and Pruning Ratio Exploration: We first evaluate the accuracy and pruning ratio by exploring different parameters α_r. For each round, we set α_r from -0.2 to 0.2 with a step of 0.1. Thus, there are five different parameters per round and 25 configurations in total. We estimate the pruning ratio and accuracy of each configuration. Since this exploration only involves inference on test sets, it takes several minutes to hours to finish. We plot all the exploration results in Fig. 10. For each scatter plot, the x-axis represents the pruning ratio, and the y-axis is the accuracy (For Task-B, the y-axis represents the

TABLE II
TOP-*k* COVERAGE ANALYSIS

Tasks	Task-A	Task-B	Task-C	Task-D
Optimal pruning ratio	11.5×	9.25×	4.77×	3.73×
Top-k coverage ratio	97.3%	91.1%	95.1%	96.0%

perplexity, lower is better). The optimal configuration should have both a high pruning ratio and accuracy.

With the exploration results, we have the following observations.

1) For NLP tasks, namely, Task-A and Task-B, we can get up to 11.5× and 9.25× pruning ratio with negligible accuracy loss (−0.48% F1-Score for Task-A and +0.17 perplexity for Task-B). On the two CV tasks, we also obtain 4.8× and 3.7× pruning with less than 0.5% accuracy loss. Moreover, even if we do not tolerate any accuracy loss, our algorithm still provides more than 2× pruning except for Task-D. Generally, the NLP tasks achieve higher pruning ratios than the CV tasks. This may be because natural language contains more redundancy than images in these tasks.
2) For Task-A, 4× pruning even improves the overall accuracy. We argue that pruning helps the model be more concentrated on those important query-key pairs, which is meaningful for question-answering tasks.
3) CV tasks are less sensitive to the pruning configurations. As we can see, Task-C and Task-D's distributions are more smooth than that of Task-A and Task-B. We argue that there are fewer "dominant tokens" for the image-classification tasks that, if wrongly pruned, will significantly degrade the accuracy (e.g., the tokens containing the question–answer for QA tasks).

We choose the configuration with the highest pruning ratio and negligible accuracy loss (within 0.5%) as our best configuration for each task and conduct the following evaluations.

Top-k Coverage Analysis: To better understand the effectiveness of our MP-MRF strategy, we also evaluate the overlap between the set of Q-K pairs selected by MP-MRF and true top-k pairs obtained by exhaustive search. As shown in Table II, we observe that under the optimal pruning ratio, the top-k coverage is higher than 95% for Task-A, Task-C, and Task-D. Even for Task-B, which has the longest sequence length, the coverage ratio reaches about 91.1%. Therefore, our MP-MRF serves as a practical approximation of top-*k* selection.

B. Architecture Evaluation

Methodology: We evaluate two configurations, namely, Energon-edge and Energon-server to meet the requirements in edge-computing and cloud-computing scenarios. The hardware and architecture parameters are listed in Table III. Energon-edge has much fewer arithmetic units compared to Energon-server and equips LP-DDR3 DRAM to reduce the power. According to our performance model in Section IV-D, the ratio of AU's parallelism m to FU's parallelism p should better be close to $[\beta/(1+\gamma)]$, where γ and β are the pruning ratios of round-0 and round-1. Our algorithm evaluation reveals that $m : p = 1 : 8$ is the most suitable for each tested task. Also, according to the estimated t_load : $t\_compute$

TABLE III
HARDWARE AND ARCHITECTURE PARAMETERS

Module	**Configurations**
Energon-edge	
DRAM	2-Channel LP-DDR3-1600, Bandwidth = 25.6GB/s;
Filtering Unit	32KB ×2 K-Buffer; 8-PE IPU;
Attention Unit	128KB ×2 K/V-Buffer; 1× MAC; 1× Softmax; 64 Multipliers for $prob \times V$;
Energon-server	
DRAM	HBM-1.0, Bandwidth = 256 GB/s;
Filtering Unit	32KB ×2 K-Buffers; 64-PE IPU;
Attention Unit	128KB ×2 K/V-Buffers; 8× MAC; 8 × Softmax; 512 Multipliers for $prob \times V$;

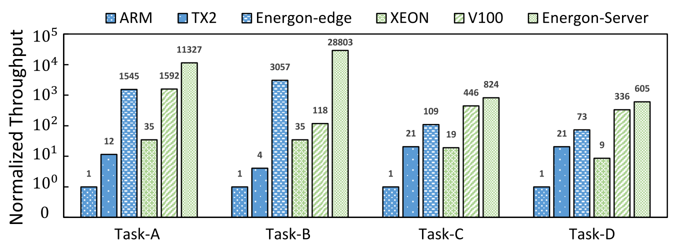


Fig. 11. Normalized throughput comparison.

ratios, we enable double-buffering on Task-A and disable half of the buffers on the remaining tasks by clock gating.

We implement Energon with Chisel [2] and compile it to Verilog RTL. We further synthesize the RLT using Synopsys Design Compiler under FreePDK 45nm standard library [35] to estimate the logic parts' area and power consumption. The power, area, and read/write bandwidth of on-chip SRAM buffers are estimated through CACTI [41]. For modeling off-chip DRAM, we simulate the memory behaviors with Ramulator [24]. According to the synthesized results, the latency of the critical path is less than 1 ns. Then, we assume the running frequency of Energon is 1 GHz. We extract each stage's actual cycles by simulating the RTL with Verilator [57], based on which a cycle-level simulator is implemented to evaluate end-to-end performance.

We compare Energon with several general-purpose computation platforms. We adopt Intel Xeon 5220 CPU [21] and NVIDIA Tesla-V100 GPU [40] as baselines to evaluate Energon-server. To evaluate Energon architecture's efficiency against mobile platforms, we also compare Energon-edge with NVIDIA Jetson-TX2 GPU [39] and ARM A72 CPU (4 cores, Raspberry Pi 4 Modle B platform [50]). We deploy the benchmarks on these platforms using Pytorch framework [44] and record the execution time of attention operations through inserting `torch.cuda.synchronize()` at the start and end points of attention layers and then calculate the spent time. We conduct the evaluation using Pytorch-1.7 with CUDA-10.2 (cudnn 7.6) and MKL-2020.2 as backends. Note that to test the latency of generation model GPT-2 (Task-B), we follow SpAtten's practice and set the initial length as 992 and measure the latency of generating 32 tokens.

Throughput Improvement: We estimate the throughput of Energon accelerators on all the benchmarks via simulation. The speedup over baseline platforms is shown in Fig. 11. As

TABLE IV
AREA AND POWER BREAKDOWN (CORE PART)

Module	Component	Area(%)	Power%
Filtering Unit	Arithmetic	2.96 | 2.22*	4.28 | 6.55*
	Buffer	7.31 | 29.13*	7.95 | 22.23*
	Control	0.14 | 0.29*	0.16 | 0.45*
Attention Unit	Arithmetic	27.14 | 9.31*	30.88 | 10.79*
	Buffer	62.21 | 58.55*	56.44 | 59.16*
	Control	0.25 | 0.50*	0.29 | 0.81*
Total		8.62 | 4.20* (mm^2)	0.89 | 0.32* (W)

* Energon-edge

TABLE V
POWER BREAKDOWN OF ENERGON

Tasks	Core Part	Memory Interface	DRAM	Total
Energon Server	0.89W	2.4W [1]	7.3W [69]	10.6W
Energon Edge	0.32W	0.9W [42]	1.5W [33]	2.7W

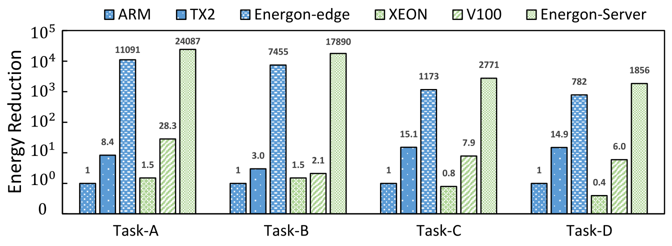


Fig. 12. Energy saving over baselines.

we can see, Energon-edge achieves 73× (Task-D) to 3057× (Task-B) speedup over ARM A72 CPU among the four tasks and achieves 3.4×(Task-D) to 764×(Task-B) speedup against TX2 GPU. Compared to Xeon CPU and V100 GPU, Energon-server shows 168× and 8.7× geo-mean speedup. Our proposed Energon architecture obtains considerable throughput improvement. Fig. 13(a) shows the speedup breakdown of Energon-edge over ARM CPU on Task-A. As we can see, the specialized hardware pipeline is 169× faster than ARM CPU. The speedup partially comes from our padding-free dataflow (2.2×). The CPU baseline running hugging-face framework cannot deal with SQuAD dataset well, whose input lengths vary a lot among samples and thus requires padding. Our MP-MRF and ODF strategies bring 8.3× and 1.1× speedup, respectively. Considering that task-A is a computation-intensive task, MP-MRF successfully reduces overall latency.

Area, Power, and Energy: We estimate the area and power of Energon and list the breakdown of the core parts in Table IV. The area and power are reported by Design Compiler using FreePDK 45-nm library [35]. The overhead of on-chip SRAMs is estimated using CACTI [41]. As we can see, Energon-edge and Energon-server utilize about 4.20 mm^2 and 8.62 mm^2 areas, respectively. For Energon-server and Energon-edge, the FUs only occupy 10.41% and 31.64% of total area, and consume 12.39% and 29.23% of total power, which demonstrates the efficiency of the proposed multiround filtering mechanism. Adopting low-bitwidth arithmetic in FU and the result-reusable mix-precision PEs jointly save the power and area. In Table V, we also list the peak power consumed by the memory interface and external DRAM.

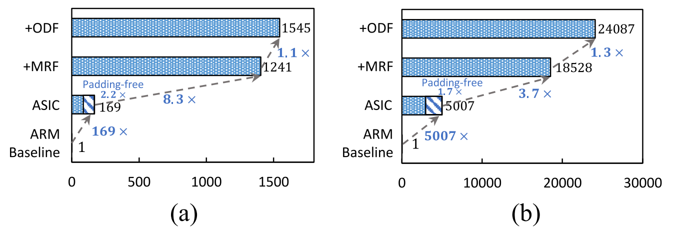


Fig. 13. Speedup and energy-saving breakdown. (a) Speedup breakdown. (b) Energy-saving breakdown.

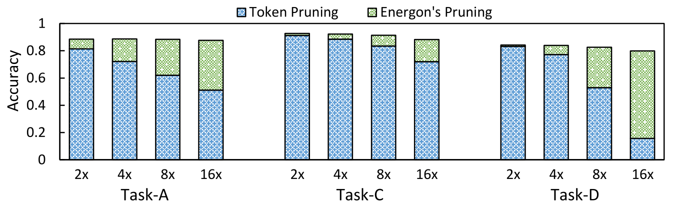


Fig. 14. Accuracy comparison with SpAtten's token pruning.

We further evaluate the end-to-end energy consumption of Energon accelerators and the baseline platforms on our benchmarks. The energy of the core parts is estimated using *latency × power*, while the energy consumed by the memory interface and DRAM is estimated according to total memory access. Specifically, we derive the energy consumption of DRAM using Ramulator and DRAMPower [5]. We profile the energy consumption of Xeon CPU with PyRAPL [45]. The running power of TX2 GPU and NVIDIA V100 GPU is estimated using PyNVML [51]. For ARM A-72@4-Cores, we estimate the power to 4 W. We plot the energy reductions on four tasks in Fig. 12. Thanks to their high throughput and low power usage, Energon-edge and Energon-server are more energy efficient by orders of magnitude (10^3×–10^4×) compared to the other general-purpose platforms. In Fig. 13(b), we also show the energy-saving breakdown. MP-MRF and ODF bring 3.7× and 1.3× energy saving, respectively. We also find that on memory-intensive tasks such as Task-B, ODF saves more energy (up to 1.5×), which is not shown in the figure.

C. Comparisons With Existing Designs

Comparison With SpAtten: SpAtten [68] adopts the cascaded token-pruning to remove "unimportant" tokens permanently. It accumulates attention probabilities among layers and prunes the tokens with low accumulative probabilities. However, such a coarse-grained pruning method suffers significant accuracy loss without retraining. We compare the MP-MRF method with token pruning on our benchmarks. For a fair comparison, we implement both methods with the same framework and adjust the pruning parameters to ensure that they have the same pruning ratio. We do not prune the rows of attention matrices when implementing token pruning, which otherwise will result in extremely low accuracy without retraining.

As we can see in Fig. 14, SpAtten shows lower accuracy on the three accuracy-oriented tasks. When the pruning ratio reaches 16×, our method's accuracy is on-average 2.32× higher than token pruning's. With the same accuracy, Energon achieves averagely 5.3× higher pruning ratio. We assume

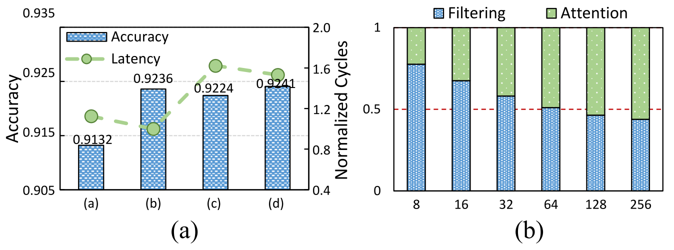


Fig. 15. Design Space Exploration. (a) Exploring filtering configurations. (b) Exploring parallelism of selector.

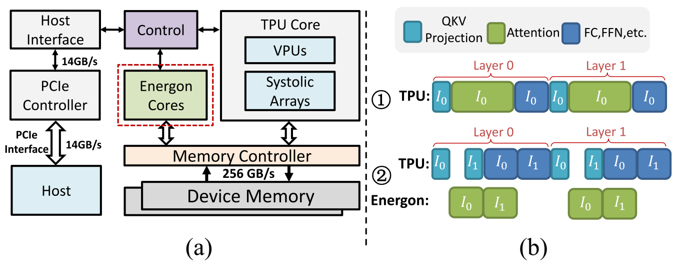


Fig. 16. (a) Conceptional integration of Energon with a TPU-like NN accelerator. (b) Execution flow comparison of ① TPU-only system and ② Energon-equipped system.

SpAtten has 2× pruning ratio on the three tasks and Energon has 16×, 8×, and 8× pruning ratios on Tasks-A, C, and D. We assume they all equip HBM-1.0 and the MACs have the same computational capacity. Since SpAtten's head pruning and progressive quantization also contribute 1.1× and 2.8× speedup, thus Energon-server should have $5.3/(1.1 \times 2.8) = 1.7\times$ higher effective throughput over SpAtten. Moreover, Energon has 1.6× higher energy efficiency, assuming SpAtten is 9.9w (7.3W HBM-1.0 + 2.6W core [68]).

Comparison With A^3: A^3 [15] adopts several numerical approximations to reduce the overall complexity. However, A^3 needs to load all the data from DRAM to perform candidate selection and postscoring selection. It consumes more energy than Energon, especially when the pruning ratio is high. For example, under the *aggressive* mode of A^3, it gets about 22× pruning ratio on the SQuAD-v1 dataset (Bert-base model). We adjust the pruning parameters of Energon and evaluate the accuracy and DRAM access reduction with the ODF strategy. Since A^3 adopts 8-bit quantization, to make the comparison fair, we also adjust the attention bitwidth to 8-bit. Our Energon accelerator achieves a sightly higher $F1$-score under the same pruning ratio (0.869 versus 0.867). Thanks to the ODF strategy, Energon saves about 34.9% of total DRAM read and achieves 1.35× total DRAM access reduction. Assuming that both accelerators equip 2-channel LPDDR3-1600 DRAM, Energon saves about 19.8% of total DRAM access energy. Moreover, we downsize Energon-edge to make it have the same amount of K and V buffer with A^3. According to our estimation, Energon saves about 1.5× energy and requires 1.2× smaller area compared to A^3. Assume A^3 and Energon have the same pruning ratio and computation ability, Energon also shows 1.25× average speedup against A^3, due to the saving of data-loading time.

D. Design Space Exploration

Filtering Rounds: To investigate the most suitable filtering rounds, we choose Task-C as a representative task and test four configurations: 1) 1-2 (denotes using 1-bit tensors in the first round and 2-bit tensors in the second round); 2) 2-4; 3) 1-2-4; and 4) 2-4-8. We evaluate their accuracy on Task-C and estimate the average cycles spent by multiround filtering. We adjust the pruning parameters of each configuration to make sure they have the same pruning ratio (4×) and the highest accuracy. The comparison results are shown in Fig. 15(a). As we can see, among the four configurations, 2) 2-4 has the lowest execution cycles and high overall accuracy. Compared to 2) 2-4, 1) 1-2 has much lower accuracy, and since the 1-bit filtering is not accurate, it has a low pruning ratio in round-0, resulting in a longer total execution time than 2). The three-round filtering spends much more cycles than the 2-round filtering, and the 3) 1-2-4 has even lower accuracy because of using 1-bit filtering in round-0. The 2-4-8 filtering has slightly higher accuracy but demands 8-bit ALUs and takes more cycles. Therefore, we argue that 2) 2-4 is an efficient choice for implementing multiround-filtering. The top-k coverage analysis in Section V-A also proves that the 2-4 configuration achieves high top-k coverage on Tasks-A, C, and D. Though it is not optimal for Task-B, for the current design, 2–4 configuration is the best selection for most of our tasks.

Parallelism of the Selector Module: Considering that the Selector module (see Section IV-B) is in the critical path of FU, it will affect the overall throughput if its parallelism (the number of parallel comparators) is not sufficient. To determine a suitable configuration, we explore a broad range of parallelism (from 8 to 256) under the settings of edge-server and test on Task-C. The ratios of total filtering cycles and attention cycles are shown in Fig. 15(b). As we can see, parallelism larger than 64 makes FU no longer the bottleneck, as FU and AU are pipelined during execution. Therefore, we adopt 64 comparators in a selector to provide sufficient performance for multiround filtering.

VI. Discussion

A. Full-Model Performance

The Energon co-processor is plugin-compatible with other NN accelerators, wherein only the attention operations are offloaded to Energon. To evaluate the full-model performance on such a heterogeneous system, we present a conceptional system and discuss its execution pipeline.

System Integration: As shown in Fig. 16(a), we demonstrate the integration of the Energon co-processor with a TPU-style [23] NN accelerator. The TPU core contains vector-processing-units (VPUs) for vector operations and systolic arrays for matrix–matrix multiplication. Multiple Energon cores are placed beside the TPU core and share the same DRAM. With the shared device memory, Energon and TPU cores can access each other's outputs without any extra data movement. We list the system configurations in Table VI. We refer to PREMA [10] for the setting of TPU parameters. We set eight Energon cores, which can process different queries in parallel. Both Energon and TPU cores execute operations according to the instructions sent from the host CPU [23].

We compare the Energon-equipped system with the original TPU-only system. Fig. 16(b) shows their different execution

TABLE VI
SYSTEM CONFIGURATIONS

TPU Core Parameters	
Systolic-array dimension	128×128
VPU dimension	SIMD-16 $\times 32$
PE operating frequency	700 MHz
On-chip SRAM size	8 (activations) + 4 (weights) MB
Energon Co-Processor Parameters:	
$8 \times$ Energon-server	
Memory system	
Memory bandwidth	256GB/s
Memory access latency	100 cycles

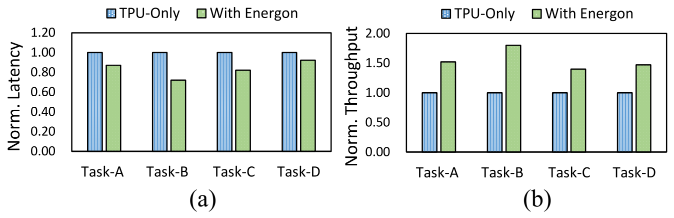


Fig. 17. End-to-end performance comparison.

flows. According to Fig. 2, we divide the operations of transformer blocks into three sequentially executed parts, namely, the QKV projection operations, attention operations, and the remaining FC/FFN/Actication operations. In the Energon-equipped system, the attention operations are executed on Energon, while the other two operations are executed by TPU. To improve the throughput, we enable pipelined execution in the Energon-equipped system. As Fig. 16(b) shows, the operations concerning two successive input sequences (I_0 and I_1) are interleaved on both TPU and Energon processors, greatly improving the system throughput.

End-to-End Performance: We evaluate the full-network performance on both systems. To achieve this, we adopt SCALE-Sim [53] to simulate the behavior of the TPU core and integrate it with our Energon simulator. For fair comparisons, we scale up the frequency of the TPU-only system and ensure that the two systems have the same tera operations per second (TOPS). Fig. 17 shows the end-to-end execution latency and throughput comparisons on the four tasks. With Energon, we can achieve 1.21× lower end-to-end latency due to the saved attention cycles. Also, by offloading attention operations to Energon cores and enabling pipelined execution, Energon-equipped can achieve on-average 1.55× higher throughput. In brief, Energon can obviously speed up the full-model network execution through dynamic sparse attention.

Scalability Evaluation: By default, we equip eight Energon cores in the system. Equipping more cores may further reduce the attention cycles. Therefore, we set different numbers of Energon cores and evaluate the scalability. As shown in Fig. 18, four Energon cores cannot outperform the TPU-only system due to the insufficient computational capacity. Setting the core numbers from 4 to 16 reduces the end-to-end latency rapidly. However, keep adding the cores will not bring obvious performance gain. On the one hand, attention operations are no longer the bottleneck. On the other hand, all cores share the memory bandwidth. Too many cores may make the system memory bounded, even with double buffers.

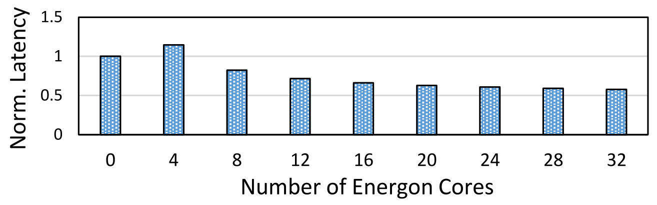


Fig. 18. Scalability exploration on Task-C. We use zero Energon cores to denote the TPU-only system.

Comparison With SpAtten: SpAtten's cascaded token pruning algorithm can also prune FC layers, which are unpruned in our settings. However, as stated in Section V-C, without model retraining, SpAtten's methods can only prune 2× attention operations, while Energon can achieve 16×, 8×, and 8× pruning on Tasks-A, C, and D. We assume that SpAtten can also prune 2× FC layers and ignore the huge accuracy loss (impossible though), then the nonattention operations of a SpAtten-equipped system should be roughly 2× faster than our Energon-equipped system, if having the same GOPS. According to Section V-C, Energon can provide 1.7× higher attention speed. Therefore, the two systems will achieve similar end-to-end performance in most cases where the cycles of attention and FC layers are close. For attention-dominated cases, Energon will show more advantages.

VII. RELATED WORK

A. NN Accelerators

There have been various accelerators designed for traditional DNNs. They accelerate convolution layers and FC layers through 1-D IPUs [6], [8], [37] or 2-D PE arrays [7], [14], [23]. These works leverage data-level parallelism to speed up the execution and also save energy through dataflow optimization. Some other works [16], [36], [43], [73], [76], [77] exploit the sparsity of DNN models to reduce computation and eliminate memory footprint. Still, some NN accelerators leverage dynamic mix precision to mitigate the computational complexity. Two recent works are DRQ [59] and DUET [30]. They adopt 4-bit tensors to compute the "unimportant" parts of the feature maps and use 8-bit or 16-bit tensors to process the "important" parts, similar to our MP-MRF mechanism. However, they are designed for convolutional layers and cannot directly support MP-MRF.

B. Efficient Transformers

Apart from the sparse attention in Section II-B, several recent works implement efficient transformers using low-bit quantization, knowledge distillation, even neural-architecture search (NAS), etc. To name a few, Q8Bert [72] adopts 8-bit quantization to avoid costly floating-point operations. GOBO [71] also proposes to compress the majority of parameters to 3 bits while maintaining their accuracy. QBert [55] and TernaryBERT further quantize the model weights to 2 bits. HAT [67] leverages neural architecture search to design an efficient transformer model for edge devices. However, these low-bit transformers still need high-precision dense attention operations to maintain accuracy. Our proposed dynamic sparse attention mechanism can further reduce the computational overhead on top of these methods.

VIII. Conclusion

In this article, we proposed Energon, a novel algorithm-architecture co-design solution to accelerate the emerging transformers with dynamic sparse attention. We presented an MP-MRF algorithm to dynamically identify important query-key pairs at runtime using low-precision tensors and only adopt high-precision tensors in the attention stage to reduce the overall complexity. We also designed an Energon co-processor architecture to accelerate such an algorithm. Elaborated pipelines and specialized optimizations jointly boost the performance and reduce energy consumption.

References

[1] A. Alzahmi, M. Alswat, and C.-C. Lin, "Energy efficient 5Gb/s/pin base-band transceiver for 3D memory interface," in *Proc. 10th Annu. Comput. Commun. Workshop Conf. (CCWC)*, 2020, pp. 827–829.

[2] J. Bachrach *et al.*, "Chisel: Constructing hardware in a scala embedded language," in *Proc. 49th Annu. Des. Autom. Conf. (DAC)*, San Francisco, CA, USA, 2012, pp. 1216–1225.

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

[4] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," 2020, *arXiv:2005.12872*.

[5] K. Chandrasekar, C. Weis, Y. Li, B. Akesson, N. Wehn, and K. Goossens. "DRAMPower: Open-Source DRAM Power & Energy Estimation Tool." 2012. [Online]. Available: URL: http://www.drampower.info

[6] T. Chen *et al.*, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proc. Architect. Support Program. Lang. Operat. Syst. (ASPLOS)*, Salt Lake City, UT, USA, 2014, pp. 269–284.

[7] Y. Chen, J. S. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proc. 43rd ACM/IEEE Annu. Int. Symp. Comput. Architect. (ISCA)*, Seoul, South Korea, 2016, pp. 367–379.

[8] Y. Chen *et al.*, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchit. MICRO*, 2014, pp. 609–622.

[9] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," 2019, *arXiv:1904.10509*.

[10] Y. Choi and M. Rhu, "PREMA: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *Proc. IEEE Int. Symp. High Perform. Comput. Architect. (HPCA)*, 2020, pp. 220–233.

[11] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

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

[13] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," 2020, *arXiv:2010.11929*.

[14] Z. Du *et al.*, "ShiDianNao: Shifting vision processing closer to the sensor," *SIGARCH Comput. Archit. News*, vol. 43, no. 3S, pp. 92–104, Jun. 2015.

[15] T. J. Ham *et al.*, "A^3: Accelerating attention mechanisms in neural networks with approximation," in *Proc. IEEE Int. Symp. High Perform. Comput. Architect. (HPCA)*, 2020, pp. 328–341.

[16] S. Han *et al.*, "EIE:Efficient inference engine on compressed deep neural network," in *Proc. 43rd ACM/IEEE Annu. Int. Symp. Comput. Architect. (ISCA)*, Seoul, South Korea, 2016, pp. 243–254.

[17] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2015, *arXiv:1510.00149*.

[18] D. Hendrycks and K. Gimpel, "Gaussian error linear units (GELUs)," 2016, *arXiv:1606.08415*.

[19] J. Ho, N. Kalchbrenner, D. Weissenborn, and T. Salimans, "Axial attention in multidimensional transformers," 2019, *arXiv:1912.12180*.

[20] Huggingface. "Huggingface Transformers." [Online]. Available: https://github.com/huggingface/transformers (Accessed: Aug. 1, 2021).

[21] Intel. "Intel Xeon Gold 5220 Processor." [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/193388/intel-xeon-gold-5220-processor-24–75m-cache-2–20-ghz.html (Accessed: Aug. 1, 2021).

[22] Jeons World. "ViT-Pytorch." [Online]. Available: https://github.com/jeonsworld/ViT-pytorch (Accessed: Aug. 1, 2021).

[23] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th Annu. Int. Symp. Comput. Architect.*, 2017, pp. 1–12.

[24] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible DRAM simulator," *IEEE Comput. Archit. Lett.*, vol. 15, no. 1, pp. 45–49, Jan.-Jun. 2016. [Online]. Available: https://doi.org/10.1109/LCA.2015.2414456

[25] N. Kitaev, L. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," in *8th Int. Conf. Learn. Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26–30, 2020*. OpenReview.net, 2020.

[26] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Rep. TR-2009, 2009.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. 26th Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, 2012, pp. 1106–1114.

[28] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "AlBERT: A lite BERT for self-supervised learning of language representations," 2019, *arXiv:1909.11942*.

[29] M. Lewis *et al.*, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proc. 58th Annu. Meeting Assoc. Comput. Linguist. (ACL)*, 2020, pp. 7871–7880.

[30] L. Liu *et al.*, "DUET: Boosting deep neural network efficiency on dual-module architecture," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2020, pp. 738–750.

[31] Y. Liu *et al.*, "RoBERTa: A robustly optimized BERT pretraining approach," 2019, *arXiv:1907.11692*.

[32] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, 2017, pp. 1–9. [Online]. Available: OpenReview.net

[33] Micron. "8Gb, 16Gb: 253-Ball, Dual-Channel Mobile LPDDR3 SDRAM." [Online]. Available: https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/mobile-dram/low-power-dram/lpddr3/253b_12–5x12–5_2ch_8–16gb_2c0f_mobile_lpddr3.pdf?rev=1b66d5710434460eb13dc3be8faa6d77 (Accessed: Aug. 1, 2021).

[34] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Proc. 11th Annu. Conf. Int. Speech Commun. Assoc.*, 2010, pp. 1045–1048.

[35] NCSU. "FreePDK45." [Online]. Available: https://www.eda.ncsu.edu/wiki/FreePDK45:Contents (Accessed: Aug. 1, 2021).

[36] S. Nelson *et al.*, "Rapid configuration of asynchronous recurrent neural networks for ASIC implementations," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, 2021, pp. 1–6.

[37] S. Nelson, S. Y. Kim, J. Di, Z. Zhou, Z. Yuan, and G. Sun, "Reconfigurable ASIC implementation of asynchronous recurrent neural networks," in *Proc. 27th IEEE Int. Symp. Asynchron. Circuits Syst. (ASYNC)*, 2021, pp. 48–54.

[38] P. Nilsson, A. U. R. Shaik, R. Gangarajaiah, and E. Hertz, "Hardware implementation of the exponential function using taylor series," in *Proc. NORCHIP*, Tampere, Finland, 2014, pp. 1–4.

[39] NVIDIA. "NVIDIA JETSON TX2." [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/

[40] NVIDIA. "NVIDIA V100 Tensor Core GPU." [Online]. Available: https://www.nvidia.com/en-us/data-center/v100

[41] H. Packard. "CACTI." [Online]. Available: https://github.com/HewlettPackard/cacti.git (Accessed: Aug. 1, 2021).

[42] R. Palmer, J. Poulton, B. Leibowitz, Y. Frans, and N. Nguyen, "A 4.3GB/s mobile memory interface with power-efficient bandwidth scaling," in *Proc. Symp. Vlsi Circuits*, 2009, pp. 889–898.

[43] A. Parashar *et al.*, "SCNN:An accelerator for compressed-sparse convolutional neural networks," in *Proc. 44th Annu. Int. Symp. Comput. Architect. (ISCA)*, 2017, pp. 27–40.

[44] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," 2019, *arXiv:1912.01703*.

[45] Powerapi ng. "pyRAPL." [Online]. Available: https://github.com/powerapi-ng/pyRAPL (Accessed: Aug. 1, 2021).

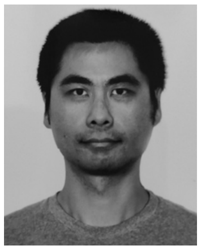
[46] J. Qiu, H. Ma, O. Levy, W. Yih, S. Wang, and J. Tang, "Blockwise self-attention for long document understanding," in *Proc. Conf. Empirical Methods Natural Lang. Process. Findings (EMNLP)*, 2020, pp. 2555–2565.
[47] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.
[48] C. Raffel *et al.*, "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.
[49] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "SQuAD: 100, 000+ questions for machine comprehension of text," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, Austin, TX, USA, 2016, pp. 2383–2392.
[50] Raspberry. "Raspberry Pi 4 Computer Model B." [Online]. Available: https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-4-Product-Brief.pdf (Accessed: Aug. 1, 2021).
[51] Rjzamora. "pyRAPL." [Online]. Available: https://pypi.org/project/pynvml. (Accessed: Aug. 1, 2021).
[52] A. Roy, M. Saffar, A. Vaswani, and D. Grangier, "Efficient content-based sparse attention with routing transformers," 2020, *arXiv:2003.05997*.
[53] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," 2018, *arXiv:1811.02883*.
[54] H. Seong, J. Hyun, and E. Kim, "Video multitask transformer network," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshops*, 2019, pp. 1553–1561.
[55] S. Shen *et al.*, "Q-bert: Hessian based ultra low precision Quantization of BERT," in *Proc. 34th AAAI Conf. Artif. Intell. (AAAI) 32nd Innovat. Appl. Artif. Intell. Conf. (IAAI) 10th AAAI Symp. Educ. Adv. Artif. Intell. (EAAI)*, New York, NY, USA, 2020, pp. 8815–8821.
[56] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–8.
[57] W. Snyder, "Verilator and SystemPerl," in *Proc. North Amer. Syste. Users' Group (NASCUG) Meeting Des. Autom. Conf.*, 2004, pp. 1–14.
[58] R. Socher, J. Bauer, C. D. Manning, and A. Y. Ng, "Parsing with compositional vector grammars," in *Proc. EMNLP*, 2013, pp. 455–465.
[59] Z. Song *et al.*, "DRQ: Dynamic region-based quantization for deep neural network acceleration," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Architect. (ISCA)*, 2020, pp. 1010–1021.
[60] A. Srinivas, T.-Y. Lin, N. Parmar, J. Shlens, P. Abbeel, and A. Vaswani, "Bottleneck transformers for visual recognition," 2021, *arXiv:2101.11605*.
[61] C. Sun, A. Myers, C. Vondrick, K. Murphy, and C. Schmid, "VideoBERT: A joint model for video and language representation learning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 7463–7472.
[62] Y. Tay, D. Bahri, L. Yang, D. Metzler, and D. Juan, "Sparse Sinkhorn attention," in *Proc. 37th Int. Conf. Mach. Learn. (ICML)*, vol. 119, 2020, pp. 9438–9447.
[63] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," 2020, *arXiv:2009.06732*.
[64] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," 2020, *arXiv:2012.12877*.
[65] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.
[66] 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. 7th Int. Conf. Learn. Represent. (ICLR)*, New Orleans, LA, USA, 2019, pp. 1–9. [Online]. Available: OpenReview.net
[67] H. Wang *et al.*, "hat:Hardware-aware transformers for efficient natural language processing," in *Proc. 58th Annu. Meeting Assoc. Comput. Linguist. (ACL)*, 2020, pp. 7675–7688.
[68] H. Wang, Z. Zhang, and S. Han, "SpAtten: Efficient sparse attention architecture with cascade token and head pruning," 2020, *arXiv:2012.09852*.
[69] Xilinx. "Virtex Ultrascale+ HBM FPGA: A Revolutionary Increase in Memory Performance." [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp485-hbm.pdf (Accessed: Aug. 1, 2021).
[70] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 5754–5764.
[71] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, "GOBO: Quantizing attention-based NLP models for low latency and energy efficient inference," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitect. (MICRO)*, 2020, pp. 811–824.
[72] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, "Q8bert: Quantized 8bit bert," 2019, *arXiv:1910.06188*.
[73] S. Zhang *et al.*, "Cambricon-X: An accelerator for sparse neural networks," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2016, pp. 1–12.
[74] W. Zhang *et al.*, "TernaryBERT: Distillation-aware ultra-low bit bert," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2020, pp. 509–521.
[75] H. Zhou *et al.*, "Informer: Beyond efficient transformer for long sequence time-series forecasting," 2020, *arXiv:2012.07436*.
[76] X. Zhou *et al.*, "Cambricon-S: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchit. MICRO*, 2018, pp. 15–28.
[77] Z. Zhou, B. Shi, Z. Zhang, Y. Guan, G. Sun, and G. Luo, "BlockGNN: Towards efficient GNN acceleration using block-Circulant weight matrices," in *Proc. 58th ACM/IEEE Des. Autom. Conf. (DAC)*, 2021, pp. 1009–1014.
[78] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable DETR: Deformable transformers for end-to-end object detection," 2020, *arXiv:2010.04159*.

Zhe Zhou received the B.S. degree from Peking University, Beijing, China, in 2019, where he is currently pursuing the Ph.D. degree in computer science.

His current research interests include computer architecture, domain-specific accelerator, and near-memory processing technology.

Junlin Liu is currently pursuing the bachelor's degree in computer science with the EECS Department, Peking University, Beijing, China.

His current research interests include machine learning systems and distributed computing.



Zhenyu Gu received the B.S. and M.S. degrees from Fudan University, Shanghai, China, in 2000 and 2003, respectively, and the Ph.D. degree from the EECS Department, Northwestern University, Evanston, IL, USA, in 2007.

His research interests include computer architecture, hardware–software co-design, and electronic design automation.

Guangyu Sun (Member, IEEE) received the B.S. and M.S. degrees from Tsinghua University, Beijing, China, in 2003 and 2006, respectively, and the Ph.D. degree in computer science from Pennsylvania State University, State College, PA, USA, in 2011.

He is an Associate Professor with the Center for Energy-Efficient Computing and Applications, Peking University, Beijing. His research interests include computer architecture, acceleration system, and electronic design automation for modern applications.

Dr. Sun is currently serving as an Associate Editor for *ACM Journal on Emerging Technologies in Computing Systems*. He is a member of ACM and CCF.