Dynamic Sparse Attention for Scalable Transformer Acceleration

Liu Liu, Zheng Qu, Zhaodong Chen, Fengbin Tu, Yufei Ding, and Yuan Xie, *Fellow, IEEE*

Abstract—Transformers are the mainstream of NLP applications and are becoming increasingly popular in other domains such as Computer Vision. Despite the improvements in model quality, the enormous computation costs make Transformers difficult at deployment, especially when the sequence length is large in emerging applications. Processing attention mechanism as the essential component of Transformer is the bottleneck of execution due to the quadratic complexity. Prior art explores sparse patterns in attention to support long sequence modeling, but those pieces of work are on static or fixed patterns. We demonstrate that the sparse patterns are dynamic, depending on input sequences. Thus, we propose the Dynamic Sparse Attention (DSA) that can efficiently exploit dynamic sparse patterns in attention. Compared with other methods, our approach can achieve better trade-offs between accuracy and model complexity. Moving forward, we identify challenges and provide solutions to implement DSA on existing hardware (GPUs) and specialized hardware in order to achieve practical speedup and efficiency improvements for Transformer execution.

Index Terms—Algorithms implemented in hardware, types and design styles, integrated circuits < B hardware, special-purpose and application-based systems, computer systems organization

1 Introduction

Transformers [1] have become the driving force for sequence modeling tasks such as neural machine translation [2], language understanding [3], and generative modeling [4], [5]. Equipped with the self-attention mechanism, Transformers are capable of handling long-range dependencies.

Despite the impressive progress made by Transformers, the computational requirements make the deployment of Transformer-based models difficult at inference time, especially when processing long sequences. The self-attention modules are the execution bottleneck under long sequences. Therefore, many studies propose Transformer variants to mitigate the quadratic time and space complexity issue. Some approaches are primary for memory footprint reduction during training while efficient inference is being understudied [6], [7], [8], [9]. Other methods use fixed or static sparse attention patterns to save computations [10], [11], [12], [13]. However, we find that intrinsic sparse patterns in attention are naturally dynamic, depending on input sequences. Thus, we propose to exploit the dynamic sparse patterns to save attention computations without sacrificing the representation power of attention. Intuitively, posing static sparsity constraints in attention could be too strong to capture dynamic attention connections.

We propose the Dynamic Sparse Attention (DSA) approach that exploits dynamic sparse patterns in attention and reduces unnecessary computations to improve efficiency. The challenge is to efficiently search for sparse patterns close to *oracle* sparse patterns that keep all the important attention weights. While contemporary studies also seek dynamic sparse patterns in attention via grouping [8], [9], [14], our work aims at inference efficiency of modeling serving rather than reducing training memory costs. We formulate the searching as a prediction problem and augment the standard attention mechanism with a prediction path. As discussed in Section 3, we first obtain an approximation of attention scores with low computational costs. Then, we predict the sparse attention patterns using the approximate attention scores. With the predicted sparse attention patterns represented as binary masks, we can save computations involved in full attention scores, $softmax$, and attention outputs.

Compared with static sparse attention methods, our method is dynamic and naturally captures sparse attention patterns of different input sequences. We observe important tokens that attract a large portion of attention weights from other tokens, similar to the global attention method [12], [13]. However, the positions of global tokens are input-dependent, and our method can effectively identify such varieties, instead of relying on domain knowledge to predetermine certain global tokens in fixed positions. Compared with other low-rank approximation methods, the approximation in DSA is only for sparsity prediction without strict and static constraints on attention positions. Therefore, our method can maintain the representation power of full attention while reducing unnecessary attention weights.

Although DSA can save theoretical computations and maintain attention capability, achieving practical speedups

• *The authors are with the University of California, Santa Barbara, CA 93106 USA. E-mail: {liu_liu, zhengqu, chenzd15thu, fengbintu}@ucsb.edu, yufeiding@cs.ucsb.edu, yuanxie@ece.ucsb.edu.*

Manuscript received 15 January 2022; revised 7 August 2022; accepted 5 September 2022. Date of publication 20 September 2022; date of current version 11 November 2022.
This work was supported in part by NSF under Grant 2124039, Use was made of computational facilities purchased with funds from the National Science Foundation under Grant OAC-1925717 and administered by the Center for Scientific Computing (CSC). The CSC was supported by the California Nano-Systems Institute and the Materials Research Science and Engineering Center (MRSEC, NSF DMR) under Grant 1720256 at UC Santa Barbara.
(Corresponding author: Liu Liu.)
This article has supplementary downloadable material available at https://doi.org/10.1109/TC.2022.3208206, provided by the authors.
Digital Object Identifier no. 10.1109/TC.2022.3208206

and energy savings on real hardware is challenging. We discuss the implications of DSA on existing GPUs and specialized hardware accelerators. We extend the fine-grained dynamic sparse patterns as searched by DSA to structural dynamic patterns, such as block-wise and vector-wise. We give the study on structural sparse patterns versus attention's expressive power and explore the opportunities for dataflow optimization and data reuse from dynamic sparsity.

Our evaluation in Section 4 shows that DSA can achieve 90-95% sparsity in attention weights without compromising model accuracy. Under this setting, the overall computational saving is up to 4.35× compared with full attention, while the sparsity prediction only introduces around 1.17% to 1.33% computational overhead. Experiments in Section 5 show that, on NVIDIA V100 GPU, applying 1×8 vector-wise sparsity of 95% sparsity ratio on DSA delivers $1.97\times$ speedup on attention module, and $1.71\times$ speedup on end-to-end Transformer inference, with only 0.12% of accuracy loss.

Finally, through hardware specialization, we explore architecture support to translate the theoretical savings to real performance speedup. We address three system-level challenges through our architecture design. First, to support large Transformer models with various configurations, we need to effectively disassemble the algorithm and identify the essential components. Prior accelerators are designed for specific components like the self-attention block [15], [16]. Instead, our design provides an efficient abstraction of the model and presents a unified architecture to support all components, achieving better area- and energy-efficiency. Besides, we further analyze different levels of parallelism on top of the proposed abstraction and present a scalable system architecture (Section 6.1). Second, low-precision computation is essential to the cost of the attention detection mechanism. To support multi-precision computations, we design a Reconfigurable Matrix Multiplication Unit (RMMU) that can be dynamically orchestrated to satisfy the throughput requirements of different computation precision (Section 6.2). Finally, when computing the attention output with the sparse attention graph, our design outperforms prior work by adopting the Token-Parallel dataflow with software-enabled workload balancing and hardware-enabled out-of-order execution.

2 Background and Motivation

Before we describe our method in detail, we first introduce the preliminaries of the standard attention mechanism used in vanilla Transformers. Then, we discuss the challenge of serving long sequences under the quadratic complexity of attention. Finally, we demonstrate that redundancy exists in attentions and dynamic sparse patterns are naturally expressed in attention.

2.1 Preliminaries of Attention

The attention mechanism is the essential component of Transformers [1]. Self-attention operates on input representations of length l, $X \in \mathbb{R}^{l \times d}$, with three linear projections namely, query, key, and value as

$$Q, K, V = XW_Q, XW_K, XW_V \tag{1}$$

, where $Q \in \mathbb{R}^{l \times d_k}$ denotes the queries, $K \in \mathbb{R}^{l \times d_k}$ denotes the keys, and $V \in \mathbb{R}^{l \times d_v}$ denotes the values. After linear projections, the attention weights $A \in \mathbb{R}^{l \times l}$ is defined as

$$A = \phi\left(\frac{QK^\top}{\sqrt{d_k}}\right) \tag{2}$$

where ϕ is the row-wise $softmax(\cdot)$ function. Finally, the output values are computed by multiplying the attention weights A with the projected values V as

$$Z = AV. \tag{3}$$

Serving Transformer-based models is challenging when the input sequence length l is large. When using long sequences, computing Eqs. (2) and (3) consumes the majority of operations and becomes the bottleneck of model evaluation. The asymptotic complexity of attention $O(l^2 d_k + l^2 d_v)$ is quadratic to sequence length l.

2.2 Intrinsic Sparsity in Attention Weights

A number of efficient Transformer variants have been proposed to mitigate the quadratic complexity of self-attention [10], [12], [13], [17]. One straightforward way to exploit the intrinsic redundancy in attention is forming sparse patterns as in

$$A = \phi(QK^\top - c(1 - M)), \tag{4}$$

where $M \in \{0, 1\}^{l \times l}$ represents the sparse attention pattern, c is a large constant $(1e^4)$ such that where $M_{ij} = 0$, indicating unimportant attention, $A_{ij} = 0$ after $softmax$ normalization. Here, we omit $\sqrt{d_k}$ for simplicity. The sparse patterns can be pre-determined into global, block, random, or a combination of different patterns. Another way to determine sparse patterns is through trainable masks. However, all these methods explore static or fixed sparse patterns, restricting viable attention connections.

2.3 Dynamic Sparse Patterns in Attention

A common motivation of sparse attention methods is that not all attention weights, i.e., probabilities, are equally important in Eq. (3). A large portion of attention weights do not contribute to attention output and are redundant. In other words, only a small portion of attention weights are useful. However, we find that sparse patterns in attention are inherently dynamic and data-dependent.

Here, we further support our hypothesis by showing the original attention weights matrix (after $softmax$ normalization) in Fig. 1. The model used here is a vanilla Transformer and the benchmark is Text Classification from Google Long-Range Arena[18]. Fig. 1 indicates that only a small amount of attention weights are with large magnitude and a significant portion is near zero. We want to emphasize that this shows the raw attention weights without forcing any sparsity constraints or fine-tuning, which indicates that redundancy naturally exists in attention. In short, attention mechanism exhibits the focused positions on a set of important tokens.

More importantly, the attention weights have dynamic sparse patterns. As shown in Fig. 1, the sparse patterns in attention weights are dynamically changing depending on the input sequence. Different heads in multi-head attention also have different sparse patterns. The characteristic of dynamic sparsity in attention weights motivates us to explore

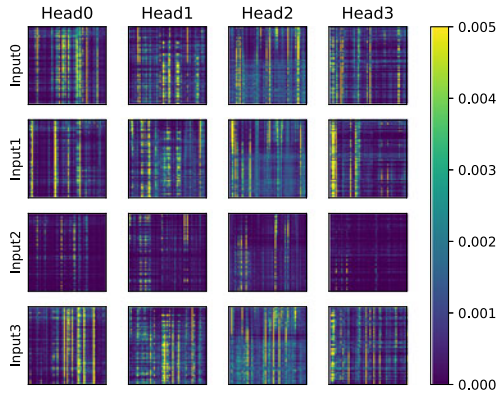


Fig. 1. Visualization of attention weights from different inputs and attention heads. Only a small amount of attention weights are important. Note values > 0.005 are clamped to show as 0.005.

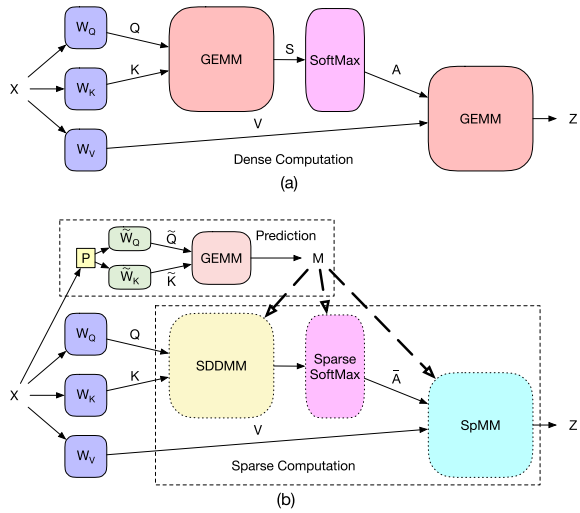


Fig. 2. (a) Standard full attention; (b) Dynamic sparse attention with approximation-based prediction and sparse computation.

effective methods to eliminate the redundancy and save computations. Prior work on static or fixed sparse patterns cannot capture the dynamically changing attention weights. Recent studies reveal similar dynamic sparse patterns in attention using modified attention [19], [20]. Instead, we promote the intrinsic dynamic sparse patterns in attention of standard transformers, and our focus is on practical acceleration of long sequences. A recent study shows that pruning near-zero attention values during inference has limited effect on accuracy [21]. The problem that our work is targeting is not only pruning unimportant attention values but also predicting which attention values to prune and saving more computations of attention scores as in Eq. (2)

3 Dynamic Sparse Patterns in Attention

From Section 2, we show that attention weights have intrinsic sparse patterns, and the positions of important attention weights are dynamically changing as different input sequences. While attention exhibits dynamic sparse patterns, how to efficiently and effectively obtain the dynamic sparse patterns remains challenging. We formulate the process of identifying sparse attention patterns as a prediction problem. The key challenge is how to obtain an approximate attention predictor that can accurately find the sparse patterns while keeping the prediction overhead small.

Here, we present *Dynamic Sparse Attention* (DSA) that exploits sparse patterns in attention weights to reduce computations. The principle is to effectively search for dynamic sparse patterns without enforcing strict and static constraints on attention while keeping the searching cost small. Our approach leverages trainable approximation to predict sparse patterns. As shown in Fig. 2, we use a prediction path based on low-rank transformation and low-precision computation. The prediction path processes input sequences functionally similar to query and key transformations but at much lower computational costs. Given the prediction results that approximate $QK^\top$ well, we can search sparse patterns based on the magnitude of prediction results.

3.1 Design of Prediction Path

We denote attention scores as $S = QK^\top$ and omit the scaling factor for simplicity. As shown in Fig. 2a, two general matrix-matrix multiplication kernels (GEMM) and one $softmax$ kernel consume the majority of computations in self-attention. We construct a pair of approximate query and key transformations in the prediction path to compute for approximate score $\tilde{S}$, as in

$$\tilde{Q}, \tilde{K} = XP\tilde{W}_Q, XP\tilde{W}_K. \tag{5}$$

Here $P \in \sqrt{\frac{3}{k}} \cdot \{-1, 0, 1\}^{d \times k}$ is a sparse random projection matrix shared by both paths, and $\tilde{W}_Q \in \mathbb{R}^{k \times k}, \tilde{W}_K \in \mathbb{R}^{k \times k}$ are parameters in approximating query and key.

Then, we have approximate attention scores $\tilde{S} = \tilde{Q}\tilde{K}^\top$. From $\tilde{S}$, we can predict sparse attention masks M using thresholds, where the threshold values are either fixed by tuning from the validation set or determined by $top-k$ searching. When $\tilde{S}$ is well approximated with accurate attention scores S, the large scores in $\tilde{S}$ are also large in S with high probability. The resulting sparse attention weights $\bar{A}$ is used to multiply the value matrix V similar to Eq. (3).

Optimization of Approximation. The random projection matrix P is constant after initialization and shared by two approximate transformations. We obtain the trainable parameters, $\tilde{W}_Q$ and $\tilde{W}_K$, through minimizing the mean squared error (MSE) as the criterion to optimize for approximation:

$$L_{MSE} = \frac{1}{B}||S - \tilde{S}||_2^2 = \frac{1}{B}||QK^\top - \tilde{Q}\tilde{K}^\top||_2^2 \tag{6}$$

where B is the mini-batch size.

Given the motivation of finding dynamic sparse patterns, the hypothesis of our method is that there exist *oracle* sparse patterns that perform well. Such that the optimization target is to approximate full attention scores S well enough to predict sparse patterns. We further give the results of applying oracle sparse patterns by directly dropping small-magnitude attention weights during inference without fine-tuning the model. As listed in Table 1, around 90% (up to 97%) of small attention weights can be dropped with negligible accuracy loss.

TABLE 1
Sparsity in Attention Weights, Where Values $<$ θ are Set to Zero

Case	Sparsity	EM	F1
Base	0%	81.49	88.70
$\theta = 0.001$	75% - 95%	81.50	88.70
$\theta = 0.01$	94% - 97%	80.51	87.85

A significant portion of attention weights that have small magnitude are redundant. The accuracy metrics are Exact Match (EM) and F1 Score.

3.2 Model Adaptation

When sparse attention scores are masked out to generate sparsity in attention, the remaining attention weights, i.e., the important weights, are scaled up as the denominator becomes small. Leaving the disturbed attention weights intact will degrade model quality. As a countermeasure, we propose to fine-tune model parameters with dynamic sparse constraints, referred to as model adaptation. With adaptation, the model evaluation accuracy can recover to be on par with full attention baselines, while the computational costs are significantly reduced.

We do not change the computational graph and the loss function of the original model, except adding dynamic sparse constraints in attention as mask M. As a result, the new attention $\bar{A}$ are sparse and only have important weights from prediction. Given a pre-trained model, our method jointly fine-tunes the model parameters and parameters of the prediction path as in

$$L = L_{Model} + \lambda L_{MSE} \tag{7}$$

where λ is the regularization factor of MSE. Our method can also train from scratch with initialized model parameters.

Our method approximates the original attention score with a low-rank matrix $\tilde{S}$. When training the model with loss function in Eq. (6), the gradient from L_{MSE} will be passed to both the low-rank approximation $\tilde{S}$ and the original attention score S. Intuitively, this loss function not only makes $\tilde{S}$ a better approximation of S, but also makes S easier to be approximated by a low-rank matrix, i.e., by reducing the rank of S. On the other hand, the loss L_{Model} guarantees the rank of S to be high enough to preserve the model accuracy. In other words, the joint optimization of L_{Model} and L_{MSE} implicitly learns a low-rank S with a learnable rank depending on the difficulty of the task. Our design brings two advantages. First, the rank of S will be automatically adjusted to tasks with different difficulty levels. Hence, our method can potentially achieve higher accuracy on difficult tasks and higher speedup on simple tasks compared with low-rank approximation methods using fixed rank. Second, as the rank of $\tilde{S}$ only implicitly influences the rank of S, the final result is less sensitive to the hyper-parameter k.

3.3 Computation Saving Analysis

DSA introduces additional computations in the prediction step, but the overall computation saving from sparse attention kernels is fruitful and can have practical speedup. The original full attention takes $O(l^2 d_k + l^2 d_v)$ MACs (multiply-and-accumulate operations) asymptotically. However, the asymptotic analysis does not consider practical concerns such as sparsity, quantization, and data reuse. Here, we augment the traditional asymptotic analysis with a sparsity factor α and a quantization factor β. In this way, DST prediction takes $O(\beta l d_k k + \beta l^2 k)$ MACs; DST attention takes $O(\alpha l^2 d_k + \alpha l^2 d_v)$ MACs. Both α and β are determined depending on tasks and underlying hardware platforms. In our settings, we choose α between 90% and 98% and our GPU kernels can achieve practical speedups. We assume the baseline model uses FP32 as the compute precision and set prediction precision to be INT4. The execution time on $softmax$ is not revealed in asymptotic analysis but is one of the major time-consuming components. Our method can also save the time of $softmax$ kernel with the same sparse attention patterns.

3.4 Implications for Efficient Deployment

Compared with standard attention, DSA exhibits two new features that can potentially affect model deployment. First, a light-weight prediction path is attached to the attention layer to search for dynamic sparse patterns. The prediction involves approximation of attention scores, which is essentially a low-precision matrix-matrix multiplication (GEMM). While NVIDIA GPUs with Tensor Cores support data precision as low as INT8 and INT4, DSA prediction can tolerate INT2 computation on certain benchmarks. Therefore, specialized hardware is preferable when seeking ultra-efficient attention estimation. In Section 6, we introduce two types of architectures to support multi-precision computations.

Second, the predicted sparse patterns can be used to reduce unnecessary attention computations. In other words, instead of computing $QK^\top$ and AV as two dense GEMM operations, we can reformulate $QK^\top$ as a sampled dense dense matrix multiplication (SDDMM) and AV as a sparse matrix-matrix multiplication (SpMM). When processing SDDMM and SpMM kernels on GPU, data reuse is the key disadvantage that limits its performance compared with GEMM. Therefore, we extend DSA to support structural sparsity that can improve the data reuse of both SDDMM and SpMM kernels. We implement customized kernels that take advantage of the sparsity locality to improve kernel performance, achieving practical runtime speedup on NVIDIA V100 GPU. Also, we demonstrate our choice of structural sparsity pattern and that DSA is able to maintain the model expressive power with the extra constraints.

As for specialized hardware, the advantage of DSA can be fully exploited as the specialized architecture and dataflow is able to deal with fine-grained sparsity, therefore achieving optimal sparsity ratio and computation reduction. However, the challenge also arises as irregular sparsity causes load imbalance and under-utilization of processing elements. Moreover, instead of independently executing SDDMM and then SpMM, we point out that more optimization opportunities can be explored when considering the whole process as a two-step SDDMM-SpMM chain. Please refer to Section 6 for more architectural design details and experimental results.

4 Algorithmic Evaluation

In this section, we evaluate the performance of DSA over representative benchmarks from Long-Range Arena [18]. We first compare the model accuracy of DSA with dense vanilla transformers and other efficient transformer models. Then, we present a sensitivity study over different configurations

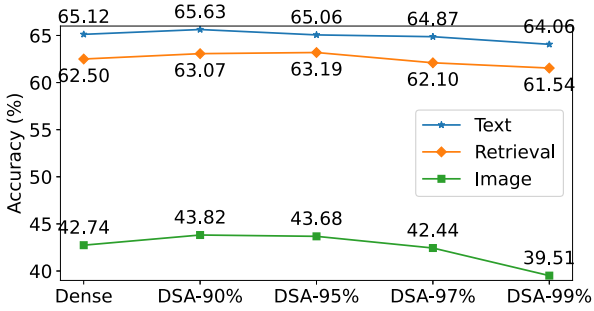


Fig. 3. Overall model accuracy of DSA (**fine-tuned** from a pre-trained checkpoint) compared with vanilla dense transformer.

of the prediction path. By choosing different number of prediction parameters, DSA is able to achieve flexible trade-offs between computational cost and model accuracy. Finally, we study the model efficiency of DSA by analyzing the computational cost (MACs) and relative energy consumption.

4.1 Experiment Settings

The datasets used are from Long-Range Arena (LRA), which is a benchmark suite for evaluating model quality under long-sequence scenarios. In LRA, different transformer models are implemented using Jax [1] API and optimized with just-in-time ($jax.jit$) compilation. We implement DSA on top of the vanilla transformer provided by LRA and compare it with other models included in LRA. Specifically, the self-attention layer in the vanilla transformer is augmented by the DSA method as described in Section 3. All the other model configurations are kept the same for a fair comparison.

We incorporate three tasks from the LRA benchmark in our experiment, including Text Classification, Document Retrieval, and Image Classification. The Long ListOps and Pathfinder tasks are excluded. We provide benchmark descriptions and experiment configurations in the supplemental material, available online.

4.2 Accuracy

Fig. 3 presents the overall model accuracy of DSA on different LRA tasks. In this experiment, the DSA model is fine-tuned from a pre-trained vanilla transformer by jointly updating the model parameters and prediction parameters using the combined loss of L_{MSE} and L_{Model}. Different percentage numbers indicate the sparsity ratio that we applied to the DSA models. For instance, *DSA-90%* means that we only keep 10% of the attention weights in each row of the attention matrix, while masking out all the other 90% of the weights. The sparsity ratio constraint is uniform for all the heads and attention layers in the DSA model.

As shown in Fig. 3, for all the evaluated tasks, dense transformer possesses a considerable amount of redundancy in the attention matrix under the long-sequence condition, which supports our previous claim in Section 2. Specifically, we can safely mask out up to 95% of the attention weights without suffering from any accuracy degradation. In fact, by jointly optimizing the model parameters to adapt dynamic sparse attention, DSA delivers slightly higher performance with 90% and 95% sparsity ratio. Even with up to 99% of sparsity, DSA still demonstrates promising performance with negligible accuracy drop compared with the dense baseline. We also compare DSA with other efficient Transformer methods on the LRA benchmark and show that DSA maintains a strong performance compared with related work. The detailed experimental setup and results can be found in Appendix B, available in the online supplemental material

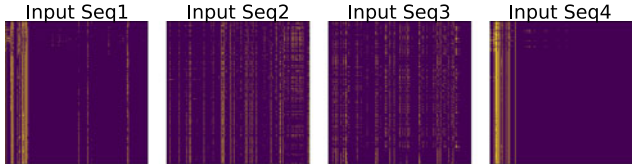


Fig. 4. *Oracle* attention mask generated by *top-k* selection.

The encouraging performance of DSA mainly comes from two aspects. First, joint optimization ensures that the DSA model can well adapt to the sparse attention patterns for computing the attention output. Second, the trainable prediction path is able to accurately capture the input-dependent patterns. Fig. 4 shows the *oracle* sparse patterns of four different input sequences obtained from *top-k* selection over the original full attention matrix. The yellow dots indicate that the important positions in the attention matrix, while the purple region is masked out. Fig. 5 shows the sparsity patterns generated by DSA prediction. As we can see from the two figures, horizontally, the sparse attention pattern changes with different input sequences. Vertically, the predicted patterns are very close to the *oracle* patterns. In our experiments, the prediction accuracy is around $85 \sim 95\%$.

To make sure the high performance of DSA comes from the proposed approach rather than the task itself, we further test two cases on the Text Classification dataset. First, we apply a 99% sparsity constraint on the vanilla transformer, but with a static local attention pattern. Second, we use a short sequence with dense attention, and let the total number of tokens in the short sequence matches with the number of important tokens in the long-sequence scenario. The results show that these two cases perform very poorly on the task, delivering a model accuracy of only 53.24% and 54.16% compared with 64.04% accuracy achieved by *DSA-99%*. This further supports our previous discussion.

4.3 Design Space Exploration of Prediction Path

One of the most important design choices of DSA is the configuration of the Prediction Path. Overall, we want the predictor to accurately capture dynamic sparse patterns. However, we also want to minimize the cost of prediction while maintaining DSA model accuracy. Thus, while we involve trainable parameters for prediction, we also introduce random projection matrix $P \in \{-1, 0, 1\}^{d \times k}$ to control the prediction parameters ($\tilde{W}_Q \in \mathbb{R}^{k \times k}, \tilde{W}_K \in \mathbb{R}^{k \times k}$), and use low-precision to

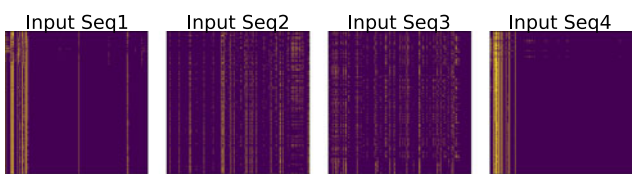


Fig. 5. Sparse attention mask generated by DSA prediction.

1. https://github.com/google/jax

TABLE 2
Change of *DSA-90%* Model Accuracy When Sweeping Random Projection Scale σ and Quantization Precision

σ	0.1	0.16	0.2	0.25	0.33	Base
DSA-90%	65.32	65.25	65.17	65.46	65.63	65.12
Precision	**Random**	**INT2**	**INT4**	**INT8**	**FP32**	**Base**
DSA-90%	60.42	64.23	65.56	65.69	65.63	65.12

reduce the computation overhead. Here, we present the sensitivity results regarding different choices of the reduced dimension size and quantization precision.

We first sweep over different sizes of k and evaluate the accuracy of *DSA-90%* on the LRA Text Classification task. Here, we use $\sigma = k/d \in (0, 1]$ to represent the size of the predictor. A Larger σ value indicates more prediction parameters and better representation power, but also larger computation overhead. As we can see from Table 2, DSA demonstrates relatively stable performance with different σ values. Even with $\sigma = 0.1$, *DSA-90%* still achieves a slightly higher accuracy compared with vanilla transformer. We believe this is because we use predictor to indicate the positions of the important attention weights, while passing the accurate attention weights to the output. Therefore, our predictor module can tolerate highly approximate computation as long as it can capture the relative importance in the attention matrix.

To further study the performance and the impact of the predictor, we conduct another experiment to sweep over different quantization precision, while fixing σ to be 0.25. As shown in Table 2, *DSA-90%* achieves good accuracy with quantized precision as low as 4-bit. Accuracy degradation occurs when the precision further scales down to 2-bit. As we go deeper into the predictor module, we collect and show the prediction accuracy in each attention block of this 4-layer DSA model. The prediction accuracy is defined by the percentage of the correct guesses among the total number of predictions. For example, for a *DSA-90%* model working on a sequence length of 2000, for each row of the attention matrix, the predictor will output 200 positions to be important. If 100 of these 200 locations actually matches with the *top-k* results, the prediction accuracy is 50%. As shown in Fig. 6, the predictor is able to maintain its prediction accuracy even with 4-bit quantization. When the precision is 2-bit, the prediction accuracy suffers a significant degradation, dropping from $60 \sim 90\%$ to $25 \sim 55\%$. Despite this, the overall model accuracy is acceptable, with only 0.89% degradation compared with the baseline transformer. We believe this is because, for the binary Text Classification task, it is more crucial to capture the very few most important attentions. Although the prediction accuracy becomes lower, the most important positions are preserved and therefore maintaining the overall model accuracy. Finally, in Fig. 6 and Table 2 we include a special case of randomly selecting 10% important positions. With this random mask applied to the model, the prediction accuracy is less than 10%, and overall model accuracy directly drops to 60.42%. This result supports our previous analysis.

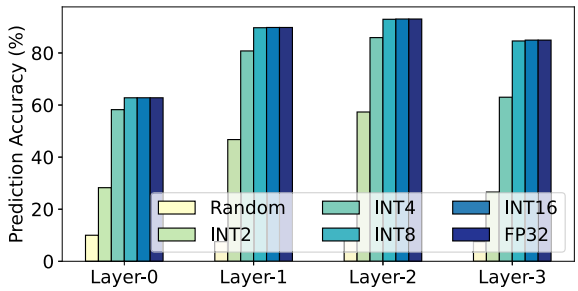


Fig. 6. The prediction accuracy of DSA in a 4-layer *DSA-90%* model with different quantization precision.

4.4 Model Efficiency

Before diving into the implementation of DSA on different hardware platforms, we first provide a theoretical analysis to illustrate the model efficiency. We start with presenting the number of required MAC operations for each attention layer. We use MAC number as the computational cost metric because the majority of the operations in the self-attention layer are matrix multiplications. We break down the total MAC operations into three parts: (1) Linear: General Matrix-matrix Multiplication(GEMM) for computing Query, Key, and Value. (2) Attention: GEMM for computing attention weight matrix and output Value. (3) Other: Other GEMMs inside the attention block like Feed-Forward layers. As we introduced earlier, the two GEMM operations in the part (2) scale quadratically with the sequence length, and we transform them to be SDDMM and SpMM in our DSA model to reduce both computation and memory consumption. Based on this setting, the computational cost breakdown of different models used in our LRA experiment is shown in Fig. 7. Comparing different tasks, the tasks with longer sequence length (Text and Retrieval) are more bounded by the Attention part. The benefit of using DSA is also more significant on the 4K tasks. Comparing within each task, it is obvious that DSA model with higher sparsity ratio delivers higher computation savings. Overall, DSA achieves $2.79 \sim 4.35\times$ computation reduction without any accuracy degradation.

Note that we do not include the computation overhead of the prediction path for generating the sparsity mask. This is because the computations conducted in prediction are in reduced precision rather than full-precision. Besides, it is inappropriate to directly project the number of low-precision MACs to the number of FP32 MACs.

Furthermore, we show the the energy consumption of DSA relative to the dense attention. We use *DSA-95%* as an example and choose $\sigma = 0.25$ and INT4 quantization. Each INT4 MAC's energy cost is projected to the relative factor of FP32 MAC, where the factor number is referenced from industry-level simulator [22] with 45nm technology. From Fig. 8 we can see that, even with the predictor overhead considered, the overall benefit is still

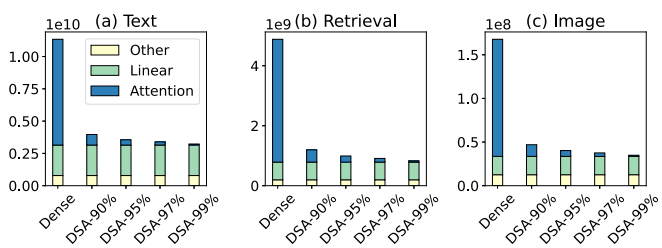


Fig. 7. Computational cost measured in the number of MACs.

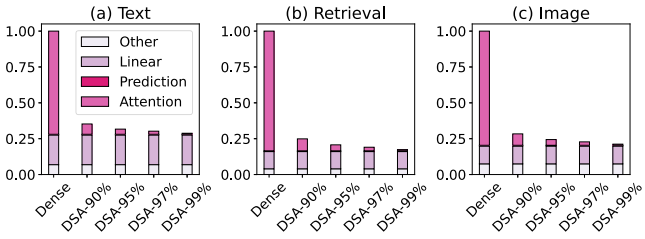


Fig. 8. Relative energy consumption projected to vanilla transformer.

compelling by virtue of the high dynamic sparsity ratio and low-cost prediction.

5 GPU Deployment of DSA

In Section 4.4, we analyze the potential of DSA in terms of reducing the total cost of Transformers. While the estimated number of MAC operations and relative energy consumption present very promising results, it remains challenging to achieve practical speedup and energy reduction on real hardware systems. In this section, we dive deeper into this problem as we discuss the implementation of DSA on GPUs. Specifically, we evaluate the challenge of mapping DSA onto GPU architectures, and demonstrate the flexibility of DSA to enable efficient algorithm-hardware co-designs.

5.1 DSA Operator Pipeline

The GPU execution pipeline of DSA is shown in Fig. 9. We first perform quantization over the input feature map X. The quantized feature map X_q is multiplied with low-precision weight matrices $\tilde{W}_Q$ and $\tilde{W}_K$ to generate approximated Query and Key matrices $\tilde{Q}$, $\tilde{K}$. We implement $\tilde{Q} * \tilde{K}$ with a low-precision GEMM kernel to generate the approximated attention weights $\tilde{S}$. Important attention connections are selected based on the values of $\tilde{S}$. The selected attention locations are represented as a CSR matrix M. Given this attention sparsity, we can reformulate $QK^\top$ as the sampled dense dense matrix multiplication (SDDMM) and AV as the sparse matrix-matrix multiplication (SpMM). Various open source sparse matrix multiplication kernels [23], [24], [25] can be leveraged to perform these two operations.

There are mainly two challenges when implementing DSA on GPU. First, the latency of attention approximation and selection needs to be carefully handled, such that it can be fully covered by the savings from sparse matrix computation. Second, due to irregular memory access and low data reuse, sparse matrix operations are difficult to significantly outperform dense matrix operations on state-of-the-art GPU systems [23]. Thus, we need to adapt algorithms to help improve kernel efficiency, while satisfying model accuracy requirements. Based on this analysis, we present the implementation details of each kernel within the execution pipeline.

5.2 Quantization and Random Projection

Normally, a quantization operator can be implemented as an epilogue of the previous kernel, such that the previous kernel directly outputs quantized results. However, since we also need the un-quantized feature map X to compute sparse attention, in DSA we fuse quantization with the subsequent kernel instead of the preceding kernel. Specifically, as shown in Fig. 9, the quantization is performed together with random projection and $\tilde{Q}, \tilde{K}$ computations. Each time we load a tile of X from global memory to GPU shared memory. After quantizing this tile, we keep it inside the shared memory to be directly used by the following operators. Therefore, we can save a round trip to the global memory for the quantized feature map.

Furthermore, as shown in Eq. (5), X_q is back to back multiplied with random projection matrix P and low-precision weight matrices $\tilde{W}_Q$, $\tilde{W}_K$. Since both P and $[\tilde{W}_Q, \tilde{W}_K]$ are fixed during inference, we can pre-compute the result of $P * [W_Q, W_K]$ and store it as a equivalent weight matrix $\tilde{W}$. Therefore, the consecutive linear transformations can be represented by a single low-precision weight matrix.

To sum up, as shown in Fig. 9, by applying pre-computation and quantization fusion, we are able to generate $[\tilde{Q}, \tilde{K}]$ using input X with just one single kernel.

5.3 Attention Approximation and Selection

As shown in Fig. 9, we compute the approximated attention weights using a low-precision dense GEMM kernel. The precision is together decided by model tolerance and hardware support. While DSA achieves no accuracy degradation with INT4 computation, Nvidia V100 GPU only offers INT8 arithmetic. On the contrary, the latter GPU architectures such as Ampere and Turing offer INT4 support, and are able to further exploit algorithm potential. After generating the approximated attention values, we perform attention selection by either comparing the results with pre-trained thresholds or by perform local top-K selection. Kernel fusion is leveraged again, as we directly do the pruning right after the results are computed and cached in shared memory. Therefore, the kernel only outputs the sparsity

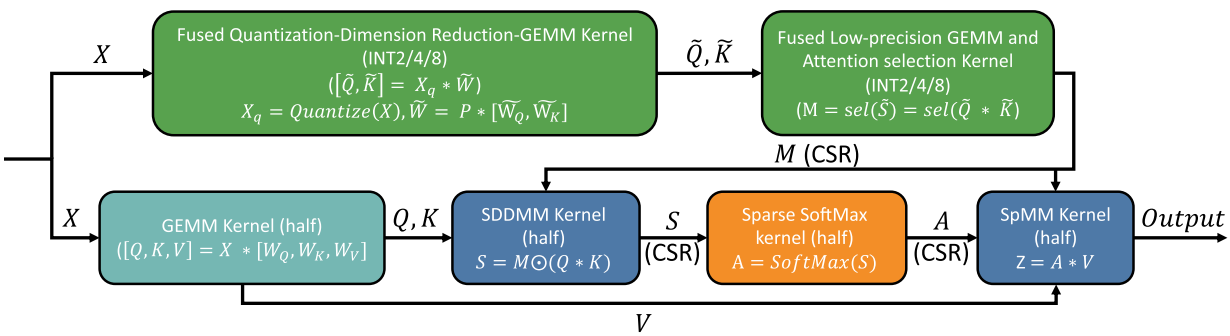


Fig. 9. Overall GPU operator pipeline of DSA.

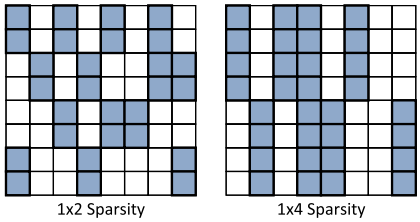


Fig. 10. Column-vector sparse encoding [23].

mask without having to write the approximate attention values.

5.4 Sparse Attention Computation

The key enabler for DSA's performance improvements is the use of sparse GPU kernels for attention computation. Specifically, we use SDDMM kernel to compute attention score, and use SpMM kernel to compute the final attention output. There are multiple open source implementations can be utilized. For example, under fine-grained sparsity and single precision, the SDDMM and SpMM kernel proposed in [26] can outperform dense GEMM kernel under $> 71\%$ and $> 90\%$ sparsity, respectively. Besides, *cusparse* [25] also achieves practical speedup at $> 80\%$ sparsity for single precision data.

However, it is far from enough to solely beat single precision GEMM operation, because inference engine is usually deployed under half (FP16) precision using advanced Tensor Core architectures. When half precision (FP16) is used for computation, above fine-grained kernels can hardly compete with GEMM kernel [23]. Consequently, the performance gain on sparse matrix multiplication can hardly mitigate the overhead of computing the prediction path in DSA. To tackle this problem, structural dynamic sparsity can be introduced to the attention selection. Specifically, instead of selecting independent attention weights, we can enforce block-wise and vector-wise constraints. Also, trade-off can be made by adjusting the block size, as larger blocks deliver higher speedup but can potentially cause accuracy loss.

In our work, we experiment on vector sparsity using the Text Classification benchmark. The performance statistics can be generalized to other benchmarks as long as a similar model configuration and comparable sparsity ratio can be guaranteed. As shown in Fig. 10, we choose column-vector sparse encoding, where the attention elements are pruned in a column-vector granularity. Column-vector sparsity provides the same data reuse as block sparsity, but its smaller granularity makes it more friendly to model training [23]. Table 3 gives the corresponding kernel speedup and model accuracy under 90% sparsity ratio. The data type is FP16 for 1×4 and 1×8 sparsity and FP32 for fine-grained sparsity. As we can see, DSA can be flexibly combined with different sparsity patterns, achieving practical runtime speedup on GPU while maintaining on-par model accuracy with full attention.

To shed some light on the results, we can trace back to the visualizations of the attention matrix in Fig. 1. As shown by the figure, despite the sparse and dynamic characteristics of the attention matrix, the distribution of important attention connections exhibits a certain degree of locality. For example, there exist some global tokens that attend to most of the tokens within a sequence. Therefore, some columns of the attention matrix will contain many important positions. Besides, local attention also indicates row-wise locality, as a token is likely to be influenced by its neighbors. Therefore, row-vector sparsity can be added to DSA for performance/accuracy exploration as well. While these fixed locality patterns have been well discussed in prior work [12], [13], DSA illustrates the dynamic distribution which motivates us to propose the prediction path to efficiently locate these important connections.

TABLE 3
Model Accuracy and Kernel Speedup Over *cuBLASHgemm*

Sparsity Pattern	vec 1×4	vec 1×8	Fine-grained
SpMM Speedup	1.57×	1.94×	1.85×
SDDMM Speedup	0.94×	1.15×	1.09×
Accuracy(%)	-0.02	-0.1	+0.5

We implement customized SDDMM/SpMM kernel for $1 \times 4/1 \times 8$ sparsity and reuse the kernel in [26] for fine-grained sparsity. Experiments are done on NVIDIA V100 GPU.

5.5 Sparse Softmax Computation

Under the long-sequence scenario, the softmax function could be a bottleneck. Let h, l, and d be the number of head, sequence length, and feature dimension of each head, respectively. Our profiling result shows that with $h = 8$, $l = 4096$, $d = 64$, softmax contributes 47% of the total execution time of the multi-head self-attention layer. By sparsifying the attention matrix, DSA directly saves both memory access and computation consumption of the softmax function to reduce execution time. We evaluate the latency of the pytorch-implemented softmax function on NVIDIA V100 GPU. Following the configuration in Text Classification Benchmark, we set batch size=16, $h = 4$, $l = 2000$ and enforce different sparsity ratios. Fig. 11 shows that the reduced softmax achieves $3.0 \sim 709.9\times$ speedup compared with dense softmax function.

5.6 Evaluation and Comparison

With the above implementation strategy, we demonstrate the overall performance of DSA on an Nvidia V100 GPU under half precision inference scheme. Specifically, we evaluate the latency and memory footprint of the attention mechanism of DSA, and compare it with other methods, including the dense vanilla Transformer and three representative efficient transformers (Performer [27], Reformer [8], and Linformer [28]). We also experiment on different sparsity ratios of DSA. Finally, we present the end-to-end

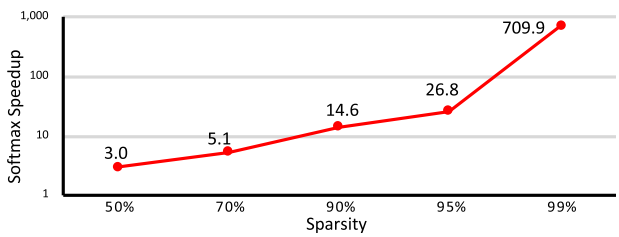


Fig. 11. Speedup of softmax with different sparsity ratios.

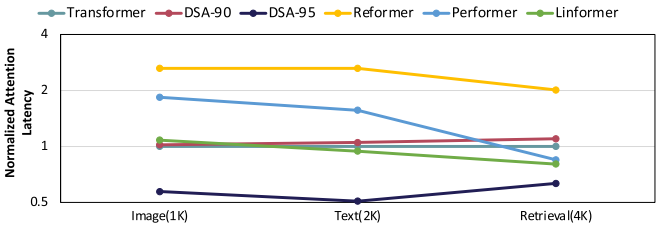


Fig. 12. Normalized GPU computation latency of different attention methods relative to the dense Vanilla Transformer. For DSA we select 1×8 attention sparsity with sparsity ratio set to 90% and 95%.

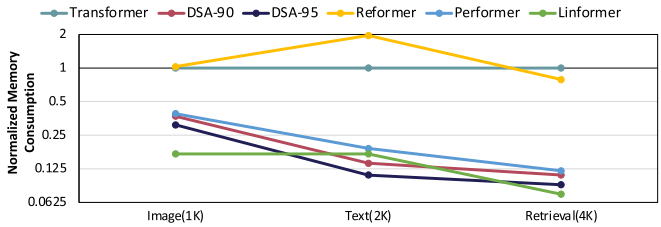


Fig. 13. Normalized memory consumption of different attention methods relative to the dense Vanilla Transformer. For DSA we select 1×8 attention sparsity with sparsity ratio set to 90% and 95%.

speedup of DSA over dense vanilla Transformers. We do not compare the end-to-end performance of DSA with other efficient Transformers, because the result is also affected by the design of non-attention modules. For DSA we are able to keep the other Transformer layers the same as dense Vanilla Transformer and therefore providing a fair end-to-end comparison.

As shown in Fig. 13, for all the evaluated benchmarks, DSA with vector sparsity is able to achieve practical speedup over dense Vanilla Transformer under a 95% sparsity ratio. The result comes from all the software-hardware co-design techniques that we proposed above, including hardware-efficient attention approximation, vector-sparsity encoding, and customized kernel design. For DSA-90, the benefit of sparse computation failed to cover the INT8 approximation overhead, but DSA still has comparable performance as dense Transformers. On the other hand, many existing efficient Transformers, despite having a linear computation complexity, cannot deliver actual wall-clock speedup due to additional overhead caused by their customized attention mechanisms. We do observe those efficient Transformers demonstrating a good scalability, indicating their potential to be applied to ultra-long sequences, such as 8K and 16K.

As for memory consumption, we collect the peak memory allocated during the attention computation, and compare the result across different models. As shown in Fig. 12, both DSA-90 and DSA-95 can achieve significant memory reduction compared with dense attention. Our kernel design plays a very important roll here. Although DSA computes a dense approximated attention, which is supposed to have a quadratic memory footprint. However, we perform attention selection right after the results is computed on-chip, without being written back to GPU global memory. Therefore, we avoid the most expensive data movement of DSA. For the original attention computation path, DSA reduces the memory consumption by using the CSR format for the involved sparse matrices.

Finally, we compare the end-to-end Transformer inference performance of DSA with dense vanilla Transformer on the Text Classification benchmark. We evaluate both 1×4 and 1×8 vector sparsity with 95% sparsity ratio. As shown in Table 4, DSA is able to achieve a $1.19 \sim 1.71\times$ speedup with less than 0.12% of accuracy degradation.

6 Hardware Specialization for DSA

While adding structural constraints can potentially benefit GPU kernel implementation, the expressive power of the model is still inevitably affected. For instance, as shown in Table 3, the 1×4 vector encoding achieves comparable accuracy with full-attention, but it is lower than the accuracy of using fine-grained sparsity under the same sparsity ratio. Thus, an alternative approach is to use hardware specialization to fully exploit the potential saving from DSA.

We present the design of DOTA, a Transformer Accelerator exploring dynamic sparse patterns, which is capable of performing scalable inference by efficiently utilizing DSA. We address three system-level challenges. First, long-sequence Transformer models involve large GEMM/GEMV computations with configurable hidden dimensions. Therefore, to effectively execute different Transformer models, we need to disassemble the algorithm and identify the essential components. We provide abstraction of the model that helps us to design a scalable and unified architecture for different Transformer layers, achieving good area- and power-efficiency. (Section 6.1). Second, apart from implementing normal precision arithmetic, DOTA also needs to support low-precision computations required by the attention detection. Instead of separately implementing all the arithmetic precision, a reconfigurable design would be preferred as it can dynamically balance the computation throughput of multi-precision computations. (Section 6.2). Finally, to efficiently compute over the detected attention graph, we should tackle the workload imbalance and irregular memory access caused by attention sparsity (Appendix C, available in the online supplemental material).

6.1 Overall System Architecture

We use Fig. 14 to illustrate the overall system architecture of DOTA, and explain how it execute a single encoder block. Running decoders can be considered as a special case of encoder with strict token dependency. As depicted by the figure, DOTA processes one input sequence at a time. Different input sequences share the same weights while requiring duplicated hardware resources to be processed in parallel. Therefore, we can scale-out multiple DOTA accelerators to improve sequence-level parallelism.

For each encoder, we split it into three GEMM stages namely Linear Transformation, Multi-Head attention, and FFN. The GEMM operations in different stages need to be

TABLE 4
End-to-End Performance Speedup

Sparsity (95%)	Self-Attention	End-to-end	Accuracy
1×4	1.23×	1.19×	-0.06
1×8	1.97×	1.71×	-0.12

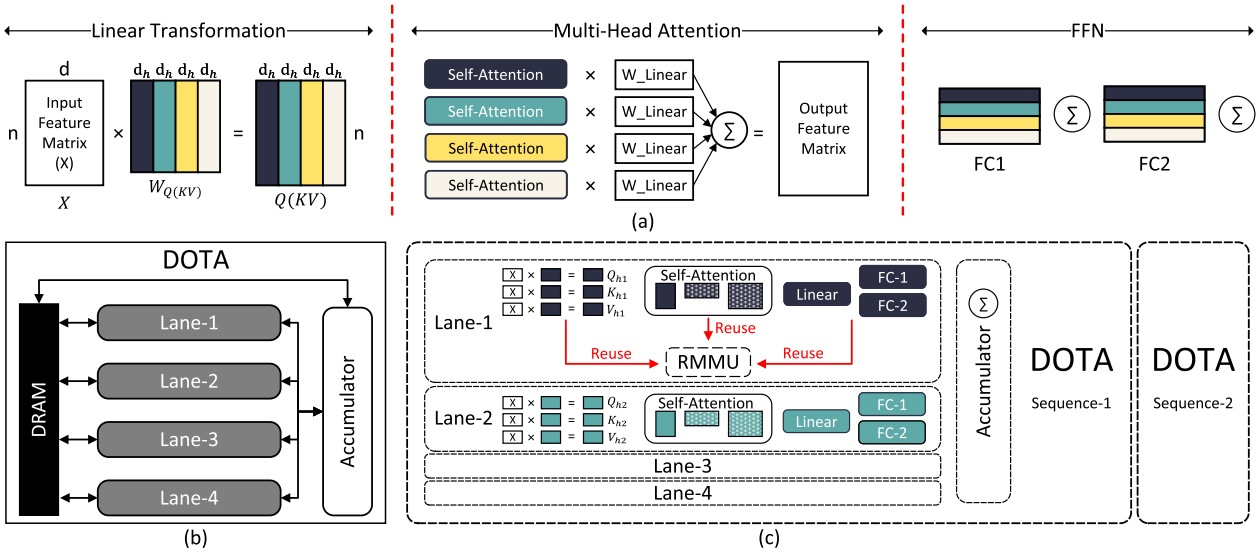


Fig. 14. DOTA system design. (a) The abstraction of a single encoder block. We divide each encoder into three sequential stages. Each stage contains multiple GEMM operations that can be further cut into chunks (represented by different colors) and mapped to different compute Lanes. (b) Overall system design of DOTA. Each compute Lane communicates with off-chip DRAM for input feature. The intermediate results are summed up in the Accumulator. (c) Computation mapping between the algorithm and hardware. Each DOTA accelerator processes one input sequence, and each Lane computes for one chunk (color).

computed sequentially due to data dependency, while each GEMM can be cut into multiple chunks and processed in parallel. Therefore, as shown by Fig. 14, we locate 4 compute Lanes in the DOTA accelerator and dedicate each Lane to the computation of one chunk. For example, during Transformation stage, each Lane contains a fraction of weight W_Q, W_K, W_V and generates a chunk of QKV. We make the chunk's size equal to the attention head size h_d. Thus, for Multi-Head Attention, each Lane can directly use the chunks previously generated by itself to compute for self-attention, keeping the data local during execution. Finally, the FC layers in the FFN stage can be orchestrated in a similar way.

Different compute Lanes share the same input at the beginning of a encoder, whereas the weights and intermediate results are unique to each Lane. Therefore, we avoid data exchanging as well as intermediate matrix split and concatenation among the Lanes. An exception of the above discussion is that, at the end of Multi-Head attention and each FC layers in FFN, we need to accumulate the results generated by each Lane. In DOTA, this is handled by a standalone Accumulator. We locate four Lanes in one DOTA accelerator because 4 is the least common multiple of the attention head numbers across all the benchmarks we evaluated. More Lanes can be implemented for higher chunk-level parallelism.

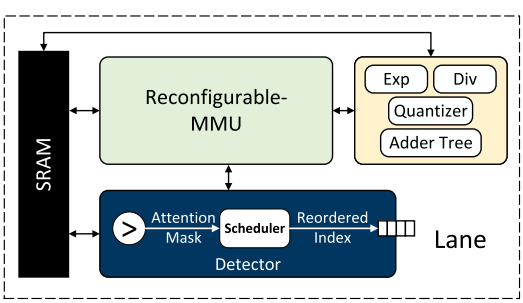


Fig. 15. Architecture of each compute Lane.

Inside each Lane, as shown in Fig. 15, there is an SRAM buffer, a Reconfigurable Matrix Multiplication Unit (RMMU), a Detector for attention selection, and a Multi-Function Unit for special operations such as Softmax and (De)Quantization. As discussed above, one large RMMU is utilized to execute all different-precision GEMM operations in each stage. Specifically, RMMU first computes low-precision (INT2/4) estimated attention score. The low-precision results are sent to the Detector to be compared with preset threshold values for attention selection. Besides selecting important attentions to be calculated later, the Detector also contains a Scheduler to rearrange the computation order of these important attention values. We incorporate this reordering scheme to achieve balanced computation and efficient memory access (Appendix C, available in the online supplemental material).

6.2 Reconfigurable Matrix Multiplication Unit Design

As presented in Fig. 15, each compute Lane contains a Reconfigurable Matrix Multiplication Unit (RMMU) which supports MAC operation in different precision. Low-precision computation occurs during the attention detection. Naively, we can support this feature with separate low-precision arithmetic units, but with the cost of extra resources to implement all supported precision levels. Besides, the decoupled design can only provide constant computation throughput for each precision, but the ratio of attention detection with respect to the other parts of the model varies from benchmark to benchmark. Thus, we need to dynamically control the computation throughput of attention detection and computation to achieve better resource utilization and energy-efficiency.

To tackle this problem, we present RMMU as shown in Fig. 16. The key idea is to design computation engine with configurable precision. As we can see from Fig. 16, RMMU is composed of a 32×16 2-D PE array, where each PE is a fixed-point (FX) MAC unit. The PE supports FX16, INT8, INT4, and

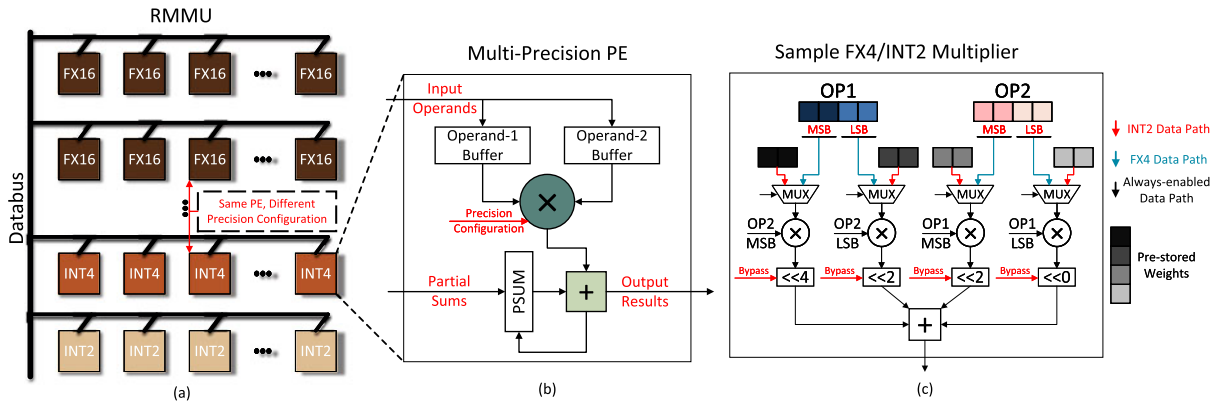


Fig. 16. Design of the Reconfigurable Matrix Multiplication Unit. (a) RMMU is composed of a 2D PE array, where each row can be configured to a specific computation precision. (b) Each PE is a multi-precision MAC unit. (c) A sample FX4/IN2 multi-precision multiplier. The key is to build up high precision multiplication data path with low precision multipliers. In low precision mode, we split and multiply the input operands with pre-stored weights and perform in-multiplier accumulation. Therefore, the computation throughput is quadratically improved while input/output bit-width are kept the same as high precision mode.

INT2 computations. FX16 is used for important attention computation and the rest are for attention detection. The RMMU can be configured to different precision at a row-wise granularity. Therefore, we can flexibly control how many rows of PE use FX16 for computation and how many rows adopt low precision to balance the computation throughput.

We design the multi-precision multiplier based on two common knowledge of computing arithmetic. First, a fixed-point multiplier is essentially an integer multiplier, only with a different logical explanation of the data. Second, we can use low-precision multipliers as building blocks to construct high-precision multipliers [29]. Without loss of generality, we present the implementation of an FX4/INT2 multiplier in Fig. 16c. As we can see, each operand is divided into MSBs and LSBs and then sent to an INT2 multiplier. A INT2 multiplier takes one fraction from each operands and generates a 4-bit partial sum. Therefore, we need four INT2 multipliers to generate all the required partial sums. The four partial sums are shifted and accumulated to give the final 8-bit result. On the other hand, if the multiplier is in INT2 computation mode, the four INT2 multipliers is able to provide four times higher computation throughput. Note that, we need 16-bit input and 16-bit output each cycle to facilitate all the INT2 multipliers. However, an FX4 multiplication only requires half the bit-width (8-bit for input/output). We address this problem by keeping half the input stationary in the multiplier, and accumulate the INT2 multiplication results before sending them out. Therefore, the input bit-width is the same as FX4 computation while the output consumes 6-bit instead of 16-bit. In other words, when working on INT2 data, we utilize the multiplier as a tiny input-stationary MAC unit which can perform 4 INT2 multiplications and accumulations each cycle.

To summarize, we implement multi-precision PEs in the RMMU and ensure scalable computation throughput when using low-precision data. Our final design implements FX-16 multiplier built up from low-precision INT multipliers as discussed above.

6.3 Scheduling Algorithm and Hardware Design

In Appendix C, available in the online supplemental material of the supplemental material, we discuss the out-of-order execution to further improve data reuse and provide the hardware design of the scheduling algorithm and details of other system architecture.

In summary, we explore token-level parallelism with software-enabled workload-balancing and hardware-enabled out-of-order execution to efficiently compute the attention output. The proposed strategy can be generalized and used in other applications with the same two-step matrix multiplication chain as shown in Eq. (8). (SoftMax is optional.)

$$O = (Q * K) * V = A * V \tag{8}$$

More importantly, even with out-of-order execution, the final result is automatically generated in a regular order. Because the irregular computation only affects the intermediate matrix A, which is completely consumed during the computation. In contrast, exploring same reordering in CNN would require a crossbar-like design to correctly store the output result [30].

6.4 Performance Speedup

We include the hardware evaluation methodology in supplementary materials, available in the online supplemental material. Fig. 17 presents the speedup of DOTA over GPUs and state-of-the-art accelerator (ELSA [16]). We evaluate both stand along attention block as well as the end-to-end performance improvements. We provide two versions of DOTA by setting the accuracy degradation of DOTA-C (Conservative) to be less than 0.5%, and limiting the degradation of DOTA-A (Aggressive) within 1.5%. As for ELSA, although it fails to reach the above accuracy requirement, we follow the original setting [16] and set the retention ratio to be 20% for performance evaluation.

As we can see, comparing with GPU, DOTA-C achieves $152.6\times$ and $9.2\times$ average speedup on attention computation and Transformer inference, respectively. On the other hand, DOTA-A achieves on average $341.8\times$ and $9.5\times$ speedups at the cost of a slightly higher accuracy degradation. The speedup mainly comes from three aspects. First, DOTA benefits from highly specialized and pipelined datapath. Second, the attention detection mechanism significantly reduces the total computations. Finally, the Token-parallel dataflow with

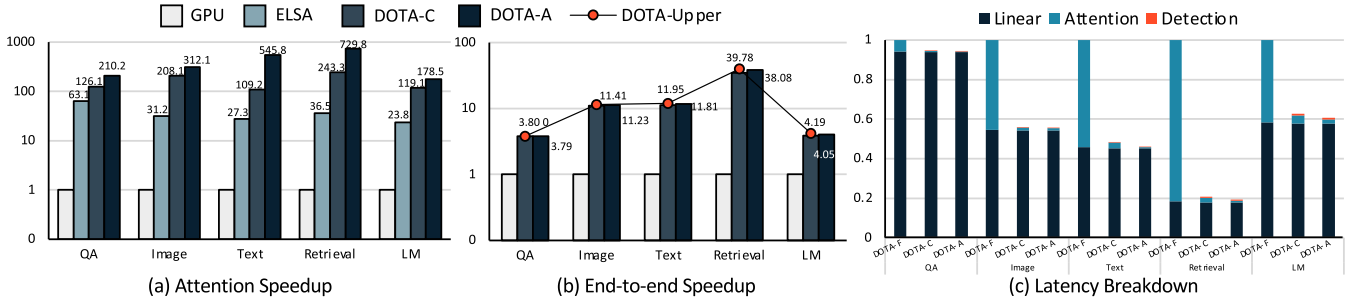


Fig. 17. (a) Speedup of DOTA over GPU and ELSA on attention block. (b) End-to-end speedup over GPU. Red dots indicate the theoretical performance upper-bound of an accelerator. (c) Normalized latency breakdown of DOTA. DOTA-F means to compute the *Full* attention graph with DOTA without detection and omission. DOTA-C (Conservative) and DOTA-A (Aggressive) both adopt attention detection, while DOTA-C allows for an accuracy degradation less than 0.5% and DOTA-A allows for 1.5%.

workload balancing and out-of-order execution further improves resource utilization. We only compare DOTA and ELSA on attention computation performance, because ELSA does not support end-to-end Transformer execution. As we can see from Fig. 17b, on average, DOTA-C is $4.5\times$ faster than ELSA and DOTA-A is $10.6\times$ faster. This improvements mainly come from higher sparsity ratio and Token-parallel dataflow.

The latency breakdown in Fig. 17c delivers two key messages. First, the latency of attention estimation is negligible compared with the overall consumption. Therefore, the Detector is both accurate and hardware efficient as we expected. Second, with the proposed detection method and system architecture, the cost of attention has been significantly reduced. The new performance bottleneck is Linear computation, which can be optimized with weight pruning and quantization. These classic NN optimization techniques can be fluently transplanted on DOTA, because our system is designed on top a GEMM accelerator with multi-precision arithmetic support and sparse computation dataflow. Overall, DOTA delivers scalable Transformer inference acceleration.

7 Related Work

Transformers with the use of self-attention mechanism are difficult to scale with sequence length because of the quadratic time and memory complexity. Our paper focuses on the exploration of sparse attention patterns in Transformers. Other orthogonal approaches such as parameters sharing [6] can mitigate the issue. We refer readers to a survey paper for a more comprehensive view of efficient Transformers [31]. Our work differentiates from these efficient Transformers as we do not simply modify self-attention mechanism to reduce complexity but with complicated impact on the representation power. Instead, we look at hardware-friendly and co-design approaches to support dynamic sparse patterns. While the Scaling Transformers [32] work does not address the long sequence attention problem, it proposes dynamic sparsity in feed-forward layers. Our work shares the similar spirit of dynamic sparsity, and we explore using dynamic sparsity in attention to address the computational complexity problem.

Static Sparse Patterns. A straightforward way to exploit attention sparsity is to set static or fixed sparse patterns, such as local windows, block-wise, dilated patterns, or a combination of static patterns [10], [11], [13]. However, as the sparse attention patterns are inherently dynamic depending on input sequences, those work lack the capability of capturing dynamic sparse patterns. As shown in our evaluation, the sparsity-saving trade-offs of representative methods using static sparse patterns are worse than our dynamic sparse attention approach.

Clustering-Based Methods. Building upon static block-sparse patterns, another line of research is to group similar tokens into chunks and perform local attention within chunks [8], [9], [14]. The similarity function used to group tokens can be hashing, clustering, or learned sorting. However, those methods are designed for training memory reduction and impractical at inference time when operating on each sequence. The quality of grouping, e.g., convergence of clustering, is not guaranteed at long sequences, and the overhead of on-the-fly clustering is not acceptable.

Approximation Methods. Recent work proposes to replace standard attention with forms of approximation of the attention weights [27], [28], [33], [34]. While we provide a comparison in our evaluation, we regard those work out the scope of our discussion for exploring sparsity in (standard) attention. Whether using a form of approximation to replace standard attention or as we suggest to predict sparse patterns explicitly is a design choice leaving up to practitioners. Our work stands from the approximation methods in three aspects. First, our method does not require retraining from scratch and incurs less fine-tuning costs. We think our method of exploiting dynamic sparse patterns is a good match to the pre-training and fine-tuning paradigm shift by transformers. Second, since we do not seek to drastically change the self-attention mechanism, our method is more flexible and adaptable to existing transformer models with scalability issues to long sequences. Besides, our method provides a run-time adjustable knob of quality versus speedup to enhance the flexibility. Lastly, from a conceptual viewpoint, our method is using efficient and hardware-friendly approximation to support dynamic sparsity as the form of approximation that is scalable to long sequences without compromising representation power.

Attention and Transformer Accelerators. Recent work adopt algorithm and hardware co-design to reduce the cost of attention mechanism. MnnFast [35] proposes to skip the computations of $A \times V$ based on the magnitude of the calculated attention scores. This method can only benefit the second GEMM of attention layer. A^3 [15] introduces attention approximation to prune the unimportant attentions. However, A^3 involves expensive online sorting, which causes significant

performance and energy overhead. ELSA [16] uses sign random projection to estimate the attention weights, making the approximation much more hardware efficient, but the model quality is hurt due to inaccurate approximation. In DSA, we address these limitations by simultaneously considering approximation accuracy and efficiency. Finally, SpAtten [36] proposes cascade token pruning and head pruning to reduce the cost of both self-attention block and subsequent layers. While removing several rows and columns of the attention matrix makes the operation regular and hardware-friendly, we find this constraint to be too aggressive as the locality of attention weights usually exists in small granularity.

Low-Precision Computing. using quantization on model parameters is a promising approach for execution efficiency, which is applicable to transformers as in recent work [37], [38]. The main challenge of our scope is mitigating the burden on attention computation, i.e., compute for attention scores and attention weights as in Eqs. (2) and (3), when serving long sequences. Note that the computations for query, key, and value as in Eq. (1) are parameterized and amenable to low-bit quantization. Instead, attention computations, which become the bottleneck when using long sequences, are non-parameterized and more sensitive to quantization errors. Even with using low-precision in attention computations, our method could still boost the performance through exploiting dynamic sparsity by predicting from low-rank transformation, set aside low-precision computation of prediction. In other words, low-precision attention computation without utilizing dynamic sparsity will still waste a significant portion of computations, even in low-precision. To summarize, our work improves the efficiency and the scalability of transformers from another perspective compared with low-precision transformers.

8 Conclusion

this paper, we present Dynamic Sparse Attention (DSA), a novel method that exploits dynamic sparse patterns in attention to reduce computational cost when serving Transformers. Specifically, we show that our method can achieve up to 95% attention sparsity without model inference quality loss. Other than prior art that uses static sparse patterns in attention, our method explores dynamic sparse patterns that are inherent in attention when processing different input sequences. Instead of replacing standard attention with other variants such as low-rank approximation methods, we augment standard attention with a prediction path as the means to locate dynamic sparsity. On one hand, attention approximation can be very efficient when only used for sparsity prediction. On the other hand, the expressive power of full attention is preserved as the important attention weights from full attention are effective in model inference. Experimental results on the LRA benchmark demonstrate superior performance and model efficiency of DSA. Furthermore, we demonstrate the potential of using DSA to improve hardware performance and efficiency. With customized kernel design and structural sparsity, DSA delivers practical speedup on GPU. The algorithm benefit can be further exploited with specialized architecture, as the hardware can fully benefit from low-precision prediction, fine-grained sparse computation, and data locality.

Acknowledgments

The first two authors contributed equally to the work.

References

[1] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6000–6010.
[2] M. Ott, S. Edunov, D. Grangier, and M. Auli, "Scaling neural machine translation," 2018, *arXiv:1806.00187*.
[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Hum. Lang. Technol.*, 2019, pp. 4171–4186.
[4] N. Parmar et al., "Image transformer," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4055–4064.
[5] T. B. Brown et al., "Language models are few-shot learners," 2020, *arXiv:2005.14165*.
[6] L. Gong, D. He, Z. Li, T. Qin, L. Wang, and T. Liu, "Efficient training of BERT by progressively stacking," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 2337–2346.
[7] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," 2019, *arXiv:1901.02860*.
[8] N. Kitaev, L. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," in *Proc. Int. Conf. Learn. Representations*, 2020.
[9] A. Roy, M. Saffar, A. Vaswani, and D. Grangier, "Efficient content-based sparse attention with routing transformers," *Trans. Assoc. Comput. Linguistics*, vol. 9, pp. 53–68, 2021.
[10] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," 2019, *arXiv:1904.10509*.
[11] J. Qiu, H. Ma, O. Levy, S. W. Tau Yih, S. Wang, and J. Tang, "Blockwise self-attention for long document understanding," 2020, *arXiv:1911.02972*.
[12] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," 2020, *arXiv:2004.05150*.
[13] M. Zaheer et al., "Big bird: Transformers for longer sequences," 2020, *arXiv:2007.14062*.
[14] Y. Tay, D. Bahri, L. Yang, D. Metzler, and D.-C. Juan, "Sparse sinkhorn attention," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 9438–9447.
[15] T. J. Ham et al., "A$\hat{3}$: Accelerating attention mechanisms in neural networks with approximation," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2020, pp. 328–341.
[16] T. J. Ham et al., "ELSA: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit.*, 2021, pp. 692–705.
[17] H. Shi et al., "Sparsebert: Rethinking the importance analysis in self-attention," 2021, *arXiv:2102.12871*.
[18] Y. Tay et al., "Long range arena: A benchmark for efficient transformers," *CoRR*, vol. abs/2011.04006, 2020, *arXiv:2011.04006*.
[19] G. M. Correia, V. Niculae, and A. F. Martins, "Adaptively sparse transformers," 2019, *arXiv:1909.00015*.
[20] H. Ramsauer et al., "Hopfield networks is all you need," 2020, *arXiv:2008.02217*.
[21] T. Ji, S. Jain, M. Ferdman, P. Milder, H. A. Schwartz, and N. Balasubramanian, "On the distribution, sparsity, and inference-time quantization of attention values in transformers," 2021, *arXiv:2106.01335*.
[22] T. Tang, S. Li, L. Nai, N. Jouppi, and Y. Xie, "NeuroMeter: An integrated power, area, and timing modeling framework for machine learning accelerators industry track paper," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, 2021, pp. 841–853.
[23] Z. Chen, Z. Qu, L. Liu, Y. Ding, and Y. Xie, "Efficient tensor core-based GPU kernels for structured sparsity under reduced precision," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2021, pp. 1–14.
[24] T. Gale, M. Zaharia, C. Young, and E. Elsen, "Sparse GPU kernels for deep learning," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage. Anal.*, 2020, pp. 1–14.
[25] M. Naumov, L. Chien, P. Vandermersch, and U. Kapasi, "Cusparse library," in *Proc. GPU Technol. Conf.*, 2010.
[26] T. Gale, M. Zaharia, C. Young, and E. Elsen, "Sparse GPU kernels for deep learning," 2020, *arXiv:2006.10901*.
[27] K. Choromanski et al., "Rethinking attention with performers," 2021, *arXiv:2009.14794*.

[28] S. Wang, B. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," 2020, *arXiv:2006.04768*.
[29] H. Sharma et al., "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit.*, 2018, pp. 764–775.
[30] L. Liu et al., "Duet: Boosting deep neural network efficiency on dual-module architecture," in *Proc. IEEE/ACM 53rd Annu. Int. Symp. Microarchit.*, 2020, pp. 738–750.
[31] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *ACM Comput. Surv.*, 2020.
[32] S. Jaszczur et al., "Sparse is enough in scaling transformers," in *Proc. Adv. Neural Informat. Process. Syst.*, 2021, pp. 9895–9907.
[33] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are RNNs: Fast autoregressive transformers with linear attention," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 5156–5165.
[34] H. Peng, N. Pappas, D. Yogatama, R. Schwartz, N. Smith, and L. Kong, "Random feature attention," in *Proc. Int. Conf. Learn. Representations*, 2021.
[35] H. Jang, J. Kim, J.-E. Jo, J. Lee, and J. Kim, "MnnFast: A fast and scalable system architecture for memory-augmented neural networks," in *Proc. 46th Int. Symp. Comput. Archit.*, 2019, pp. 250–263.
[36] 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. Archit.*, 2021, pp. 97–110.
[37] A. Bhandare et al., "Efficient 8-bit quantization of transformer neural machine language translation model," 2019, *arXiv:1906.00532*.
[38] 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. IEEE/ACM 53rd Annu. Int. Symp. Microarchit.*, 2020, pp. 811–824.
[39] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proc. 49th Annu. Meeting Assoc. Comput. Linguistics: Hum. Lang. Technol.*, 2011, pp. 142–150.
[40] D. Radev, P. Muthukrishnan, and V. Qazvinian, "The ACL anthology network corpus," *Lang. Resour. Eval.*, vol. 47, pp. 919–944, 2009.
[41] 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.*, 2016, pp. 2383–2392.
[42] Y. Tay et al., "Long range arena: A benchmark for efficient transformers," in *Proc. Int. Conf. Learn. Representations*, 2021.
[43] A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proc. 49th Annu. Meeting Assoc. Comput. Linguistics Hum. Lang. Technol.*, 2011, pp. 142–150.
[44] 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, 2019, Art. no. 9.
[45] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," 2016, *arXiv:1609.07843*.

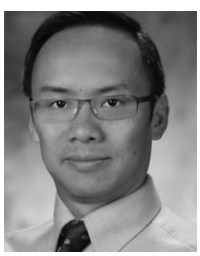
Liu Liu received the BE degree with the University of Electronic Science and Technology of China, the MS degree in electrical and computer engineering with UCSB, and PhD degree in computer science with the University of California, Santa Barbara. He has been with the Department of Electrical, Computer, and Systems Engineering at Rensselaer Polytechnic Institute (RPI) as an assistant professor since, July 2022. His research interests reside in the intersection between computer architecture and machine learning.

Zheng Qu received the BS degree from Tsinghua University, Beijing, China, in 2018. He is currently working toward the PhD degree with the Scalable Energy-efficient Architecture Lab (SEAL), Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA, USA. His research mainly focuses on algorithm-hardware co-design for emerging machine learning applications.

Zhaodong Chen received the BE degree from Tsinghua University, China, in 2019. He is currently working toword the PhD degree in the Department of Electrical and Computer Engineering, University of California, Santa Barbara. His research interests include deep learning and computer architecture.

Fengbin Tu received the BS degree in electronic science and technology from the Beijing University of Posts and Telecommunications, Beijing, China, in 2013, and the PhD degree in electronic science and technology from Tsinghua University, Beijing, China, in 2019. He is currently a postdoc researcher at the Scalable Energy-efficient Architecture Lab (SEAL), Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA. His research interests include computer architecture, deep learning, reconfigurable computing, and in-memory computing.

Yufei Ding received the BS degree and the MS degree in physics from the University of Science and Technology of China and the College of William and Mary, respectively, and PhD degree in computer science from North Carolina State University. She is joined the Department of Computer Science, University of California, Santa Barbara as an assistant professor, in Nov. 2017. Her research interests lie in the broad fields of domain-specific language design, architecture and compiler optimization, and hardware acceleration.



Yuan Xie (Fellow, IEEE) received the BS degree from Tsinghua University, and the MS and PhD degrees from Electrical Engineering Department, Princeton University. He was with IBM Microelectronics Division 's Worldwide Design Center, Pennsylvania State University, AMD Research, and UCSB. He is a Fellow of ACM, and a Fellow of AAAS, and a recipient of NSF CAREER Award and IEEE Computer Society Edward J. McCluskey Technical Achievement Award, in 2020.