

Designing Spatial Architectures for Sparse Attention: STAR Accelerator via Cross-Stage Tiling

Huizheng Wang, Taiquan Wei, Hongbin Wang, Zichuan Wang, Xinru Tang, Zhiheng Yue, *Student Member, IEEE*, Shaojun Wei, *Fellow, IEEE*, Yang Hu, *Senior Member, IEEE*, Shouyi Yin, *Fellow, IEEE*

Abstract—Large language models (LLMs) rely on self-attention for contextual understanding, demanding high-throughput inference and large-scale token parallelism (LTPP). Existing dynamic sparsity accelerators falter under LTPP scenarios due to stage-isolated optimizations. Revisiting the end-to-end sparsity acceleration flow, we identify an overlooked opportunity: cross-stage coordination can substantially reduce redundant computation and memory access. We propose STAR, a cross-stage compute- and memory-efficient algorithm–hardware co-design tailored for Transformer inference under LTPP. STAR introduces a leading-zero-based sparsity prediction using log-domain add-only operations to minimize prediction overhead. It further employs distributed sorting and a sorted updating FlashAttention mechanism, guided by a coordinated tiling strategy that enables fine-grained stage interaction for improved memory efficiency and latency. These optimizations are supported by a dedicated STAR accelerator architecture, achieving up to $9.2\times$ speedup and $71.2\times$ energy efficiency over A100, and surpassing SOTA accelerators by up to $16.1\times$ energy and $27.1\times$ area efficiency gains. Further, we deploy STAR onto a multi-core spatial architecture, optimizing dataflow and execution orchestration for ultra-long sequence processing. Architectural evaluation shows that, compared to the baseline design, Spatial-STAR achieves a $20.1\times$ throughput improvement.

Index Terms—Transformer, attention sparsity, FlashAttention, top-k, tiling, distributed attention, spatial architecture.

I. INTRODUCTION

EMPOWERED by *self-attention*, large language models (LLMs) have revolutionized fields such as chatbots [1] and code generation [2]. The *self-attention* processes three matrices: \mathbf{Q} (query), \mathbf{K} (key) and \mathbf{V} (value). First, the attention matrix $\mathbf{A} \in \mathbb{R}^{S \times S}$ is computed by $\mathbf{Q} \times \mathbf{K}^T$, where S denotes the sequence length. The resulting matrix \mathbf{A} is then passed through a softmax function for normalization before being multiplied by \mathbf{V} to generate the final output.

This work was supported in part by the National Science and Technology Major Project under Grant 2022ZD0115200; in part by the NSFC under Grant 62125403, Grant U24A20234, Grant 92464302 and Grant U24B20164; in part by the Beijing S&T Project Z251100008425010; in part by Shanghai Municipal Science and Technology Major Project; the Natural Science Foundation of Jiangsu Province Basic Research Program under Grant BK20243042; in part by the Beijing National Research Center for Information Science and Technology; in part by the Northern IC Technology Innovation Center (Beijing) Co., Ltd under Grant QYJS20232801B; and in part by the Beijing Advanced Innovation Center for Integrated Circuits. An earlier version of this paper was presented at the IEEE 57th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2024 [DOI: 10.1109/MICRO61859.2024.00093]. (Corresponding author: Yang Hu, email: hu_yang@tsinghua.edu.cn).

Huizheng Wang, Taiquan Wei, Hongbin Wang, Zichuan Wang, Xinru Tang, Zhiheng Yue, Shaojun Wei, and Yang Hu are with the School of Integrated Circuits, Tsinghua University, Beijing, 100084, China.

Shouyi Yin is with the School of Integrated Circuits, Tsinghua University, Beijing, 100084, China, and Shanghai AI Lab, Shanghai, 200232, China.

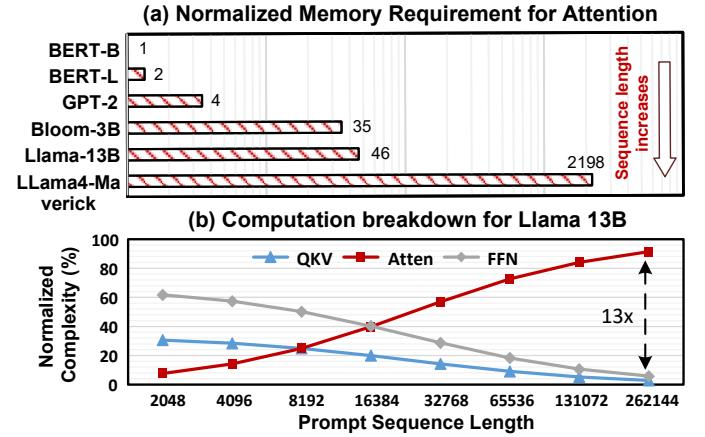


Fig. 1: (a) Normalized memory requirement for attention. (b) Computation breakdown for the Llama 13B.

LLMs increasingly demand faster inference and higher throughput, particularly for long-context tasks. However, unlike the linear complexity of $O(SH^2)$ in the feed-forward network (FFN), where H is the hidden dimension, the quadratic complexity $O(S^2H)$ of *self-attention* severely hinders the efficiency of LLMs on long sequences. From early models such as BERT [3] to recent ones like LLaMA 4-Maverick [4], the maximum sequence length has expanded over $32\times$ (512 to 16k), whereas the hidden dimension H has increased only $10\times$ (768 to 8k). This dramatic growth of sequence length results in more than a $2000\times$ increase in attention memory footprint, as depicted in Fig. 1 (a), creating significant barriers to deploying LLMs across both cloud and edge environments.

Moreover, the quadratic computation of self-attention emerges as a critical bottleneck for fast inference. As shown in Fig. 1 (b), when the sequence length reaches 16k tokens, attention surpasses the FFN as the most computation-intensive module. At 26k tokens, its cost escalates to nearly $13\times$ that of the combined QKV and FFN computations. These results underscore the pressing need to jointly optimize both computation and memory in self-attention.

Owing to linguistic redundancy, not all contexts exhibit strong correlations, leaving attention with dynamic, data-dependent sparsity. To exploit it, *Dynamic sparsity (DS) acceleration* [5–11] speculate vital Q-K pairs during runtime and calculate attention based only on these vital pairs, to evict unimportant computation. As depicted in Fig. 2, DS consists of three stages: (1) *Pre-compute stage*, estimating the attention matrix $\hat{\mathbf{A}}$ with low-precision arithmetic, e.g., 4-bit MSB. (2)

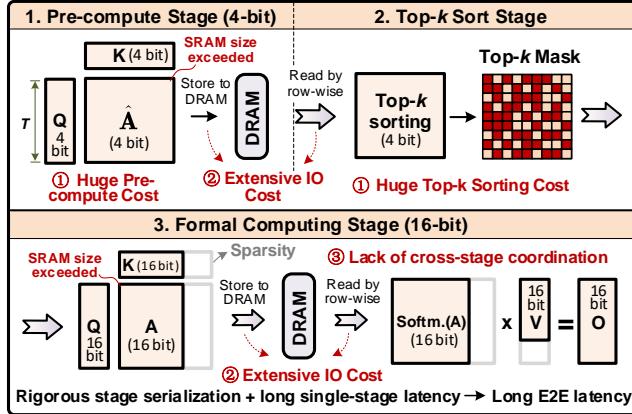


Fig. 2: Challenges for scaling existing DS works to LTPP.

Top-k stage, retaining the most important k Q-K pairs from each row. (3) *Formal computing stage*, executing attention on the retained pairs with higher precision (e.g., 16-bit).

Challenges: Limited on-chip SRAM confines most DS approaches to single-token serial processing, while modern LLMs demand support for ultra-long sequences. Large-scale token parallel processing (LTPP) reduces latency via parallelism and data reuse, necessitating DS accelerators capable of handling LTPP efficiently. However, scaling to LTPP presents three major challenges, as illustrated in Fig. 2.

a) *Prohibitive Computation Overhead*: The prediction stage (*pre-compute stage + top-k stage*), requires each query to multiply with the full-size K and perform sorting, without any sparsity reduction. For large token counts, the newly introduced stage incurs substantial computational and memory costs, which potentially offset the gains of sparsity acceleration. A promising opportunity lies in transforming computation into the log-domain to eliminate costly multiplication.

b) *Unaffordable IO Cost & Latency*: The row-wise dependencies of *top-k sorting* and *softmax* require completing an entire row from the preceding matrix. This causes frequent off-chip DRAM accesses for intermediate data, becoming a major obstacle for current accelerators. As shown in Fig. 3, the memory access time (MAT) of two state-of-the-art (SOTA) DS accelerators rises sharply with token parallelism, averaging 72% of total latency and becoming the main performance bottleneck. A potential solution is to break down the row-wise dependency of top- k sorting and softmax, by exploiting data locality and applying equivalent mathematical transformations.

c) *Uncoordinated workflow*: Existing DS accelerators lack cross-stage coordination, missing the chance to simplify later-stage operations through early-stage guidance. Despite FlashAttention (FA) [12] pioneering a tiling framework for *softmax* to reduce memory access, it incurs extra computations from repeated exponentiation and comparison needed to update MAX values across tiles. Fortunately, we identify an opportunity to simplify FA by utilizing top- k information from the prediction stage. These challenges highlight the pressing need for more advanced DS strategies under LTPP scenarios.

To this end, this paper presents STAR, a cross-stage, synergic tiling algorithm-hardware co-design for attention opti-

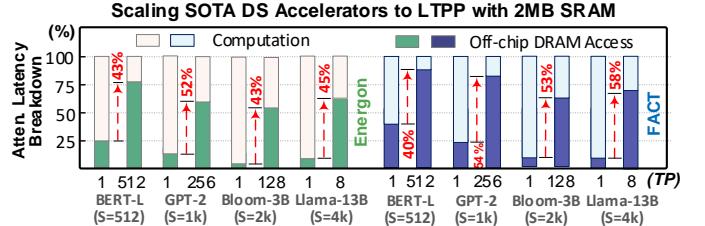


Fig. 3: Latency breakdown for SOTA DS accelerators (FACT [9], Energon [11]) with diverse token parallelism (TP).

mization in LTPP scenarios. The contributions include:

(1) The computation overhead in the *pre-compute* stage is mitigated by a multiplier-free *differential leading zero scheme* (DLZS), which reduces sparsity prediction overhead.

(2) A *sphere-search-aided distributed sorting* (SADS) strategy, which divides a long segment into sub-segments for individual sorting, effectively lowering comparison counts while enabling the tiling execution for top- k stage.

(3) A *sorted updating FlashAttention* (SU-FA), which decouples softmax row dependencies to enable tiling in the formal computation stage, while utilizing sorting information from the top- k stage to reduce computation overhead.

(4) A tailored accelerator is defined to support the above optimizations. Evaluated on 20 benchmarks, STAR achieves an average energy efficiency of 7183 GOPS/W, which is 71.2 \times and average 8.6 \times higher than Nvidia A100 GPU and three SOTA accelerators, respectively.

(5) Based on STAR, we design a spatial architecture for distributed processing of ultra-long sequences. We propose a dedicated dataflow, *DRAccelerator*, to minimize communication overhead, and a logic-level communication algorithm, *MRCA*, to efficiently implement it under 2D mesh topology. The notations used throughout this paper are listed in Table I.

II. BACKGROUND AND MOTIVATION

A. Preliminaries for Transformer

Fig. 4 illustrates the Transformer architecture and its key components. Initially, the input sequence of S tokens is mapped into an embedding matrix $\mathbf{X} \in \mathbb{R}^{S \times H}$, where H is the hidden dimension. Then, this input matrix \mathbf{X} is projected into Q, K and V spaces, each of dimension $\mathbb{R}^{S \times H}$. Next, Q, K and V are split into N_h chunks, with each chunk having a hidden dimension of H/N_h , i.e., d_h . These chunks are sent to the multi-head Attention (MHA) part, where the Q and K are multiplied to generate an attention matrix $\mathbf{A} \in \mathbb{R}^{S \times S}$, which

TABLE I: List of notations used in this paper.

Notation	Description
$S; H$	Sequence length; Hidden dimension
$N_h; d_h$	Number of heads; Hidden dimension of each head
T	Number of queries to be processed in parallel
$T_r; T_c$	Tile number along the row/column dimension
$B_r; B_c$	Tile size along the row/column dimension
$\mathbf{Q}; \mathbf{K}; \mathbf{V}$	Query, Key and Value matrices
$\mathbf{A}; \hat{\mathbf{A}}$	Attention and estimated attention matrices

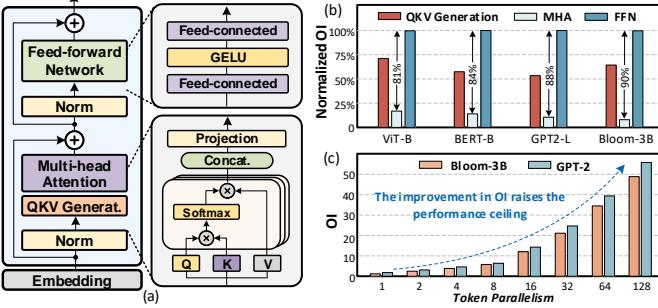


Fig. 4: (a) Transformer model. (b) Operation intensity (OI) of diverse modules. (c) OI of MHA versus token parallelism.

represents the correlation of each Q-K pair. The attention matrix is then normalized by the softmax operation as shown in Eq. (1). The softmax is applied in row-wise, and subtracting $\max_j(A_{i,j})$ from each row ensures numerical stability.

$$\text{Softmax}(A_{i,j}) = \frac{e^{A_{i,j} - \max_j(A_{i,j})}}{\sum_{k=1}^S e^{A_{i,k} - \max_j(A_{i,j})}}, j = 1, 2, \dots, S \quad (1)$$

This normalized attention is multiplied with the V activations, resulting in a matrix $\mathbf{O} \in \mathbb{R}^{S \times d_h}$, as depicted in Eq. (2). Next, the outputs from all the attention heads are concatenated and projected by a weight matrix $\mathbf{W}_O \in \mathbb{R}^{H \times H}$. Finally, the FFN with two fully connected layers generates the final outputs.

$$\mathbf{O} = \text{softmax}\left(\mathbf{Q}\mathbf{K}^T / \sqrt{d_h}\right) \times \mathbf{V}, \quad d_h = H/N_h. \quad (2)$$

We analyze the operational intensity (OI) [13] of Transformer components, which reflects the intrinsic data reuse potential. As shown in Fig. 4(b), MHA exhibits the lowest computational intensity, only about 15% of that of FFN. This suggests MHA demands significantly higher IO traffic per operation. Fig. 4(c) further shows that increasing token parallelism enhances the OI of MHA in both Bloom and GPT-2 models, as the Key matrix can be reused across more queries.

B. Prohibitive Complexity for FlashAttention (FA)

To mitigate the IO overhead of attention, the FA series [12, 14] adopt a tiling strategy that avoids materializing the full attention matrix on chip, thereby reducing costly SRAM–HBM data transfers. However, despite improved I/O efficiency, tile-wise incremental processing introduces substantial computational overhead. To quantify this, we profile the increased operations versus sequence length S , by fixing the tile size at $B_c=16$, i.e., $T_c=S/16$. As depicted in Fig. 5 (b), when $S=2048$, FA-2 consumes 8 million more exponentiations and 0.3 million more comparisons than the vanilla baseline. Since each non-matmul FLOP is roughly 16× more costly than a matmul FLOP [14], this extra overhead severely impedes attention efficacy gains.

Further, Fig. 5(c) presents the increase in computation complexity after aggregating all operations. The total complexity, which includes different operation types, is unified by

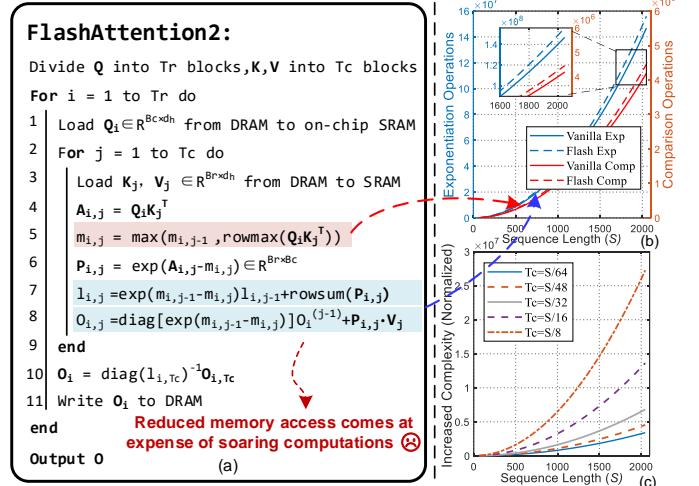


Fig. 5: FlashAttention-2 (FA2) and its computation overhead.

normalization to equivalent additions¹. As can be seen, the increased complexity escalates rapidly with increasing sequence length, with the extent of increase strongly correlating with the number of tiles T_c . A larger T_c results in a more pronounced rise in complexity, driven by the higher frequency of redundant operations across tiles, as illustrated in lines 5-8 of Fig. 5(a).

C. Dynamic Sparsity in Transformer Attention

While *self-attention* is designed to capture token correlations within a context, not all tokens exhibit strong correlations. For example, tokens like articles ‘a’ or ‘the’ contribute negligibly to semantics and yield near-zero attention values. The subsequent *softmax* would further suppress these small elements to near-zero. Therefore, their corresponding K/V vectors have a negligible impact on the attention output \mathbf{O} and can be pruned with minimal performance loss.

To exploit this runtime sparsity of attention, a large body of DS accelerators [5–11] has been proposed. Unlike static pruning techniques where the sparsity pattern is fixed offline, dynamic sparsity requires on-the-fly decisions during inference. The fundamental principle is to dynamically predict the most relevant Q–K pairs at runtime using an efficient, low-overhead mechanism and perform the high-precision attention computation only on the selected pairs. The general flow of DS is illustrated in Fig. 2.

III. MOTIVATION

A. Challenges of LTPP with Current DS Accelerators

In recent years, LLMs have seen an exponential increase in context length, from BERT’s 512 tokens in 2018 [3] to GPT-4’s 32k tokens [2], recently. This trend highlights the need for efficiently supporting LTPP. However, three key challenges lie in scaling current DS accelerators to LTPP scenarios:

(1) **Prohibitive sparsity prediction overhead.** When T queries are processed in parallel, the precomputation

¹We normalize the total computation complexity of attention with equivalent additions: $C_{\text{attention}} = \alpha \cdot N_{\text{add}} + \beta \cdot N_{\text{mul}} + \gamma \cdot N_{\text{cmp}} + \delta \cdot N_{\text{div}} + \epsilon \cdot N_{\text{exp}}$, where $\alpha = 1$, $\beta = 3$, $\gamma = 1$, $\delta = 8$, $\epsilon = 25$, according to the [15].

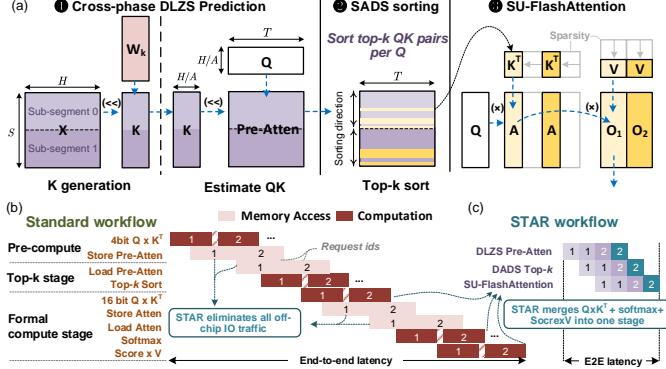


Fig. 6: End-to-end tiling pipeline workflow of STAR. (b) Comparison with the current standard DS workflow.

and sorting complexities scale as $\mathcal{O}(TSd_h)$ and $\mathcal{O}(TS^2k)$, where $k \in [0, 1]$ denotes the top- k ratio. Specifically, the pre-computation stage involves multiplying the Q matrix $\in [T, d_h]$ with the K matrix $\in [d_h, S]$, resulting in a complexity of $\mathcal{O}(TSd_h)$. In the top- k stage, the attention matrix $\in [T, S]$ is generated, where each row selects the top $S \cdot k$ elements. Since selecting each element requires $\mathcal{O}(S)$ operations, the overall comparison and selection process incurs $\mathcal{O}(TS^2k)$ complexity, resulting in significant runtime overhead. For example, applying DS to LLaMA-7B with $T=512$ and $k=0.25$ incurs over 2.6×10^8 FLOPs and 2.1×10^9 comparisons for sparsity prediction. This results in nearly $12 \times$ higher power overhead than the formal computation stage, when implemented on 45 nm CMOS at 500 MHz frequency. In addition, sparsity prediction accounts for around 57% of the total attention latency, significantly limiting LLM inference efficiency.

(2) Excessive IO overhead due to the rigidity of operator dependencies. Due to the row-wise nature of the top- k and softmax operators, directly increasing token parallelism leads to intermediate data exceeding the on-chip SRAM capacity. As a result, the data have to be offloaded to off-chip DRAM, incurring substantial IO transfer overheads. Taking CMOS 45 nm for an illustration, the energy cost for a DRAM access is around 5-20 pJ/bit, which is orders of magnitude higher than on-chip SRAM (0.1 pJ/bit). In terms of bandwidth, off-chip memory (DDR4, 25.6 GB/s) typically exhibits two orders of magnitude lower bandwidth than on-chip memory (SRAM, 19TB/s). A direct but crude approach is to enlarge the on-chip SRAM capacity, but this would damage area efficiency. For example, when deploy Bloom7B with $T=512$, it requires a substantial 5MB of SRAM, resulting in 5.72 mm² footprint under TSMC 28nm. This is $7.4 \times$ and $8.9 \times$ larger than the total area of SOTA ELSA [6] and SpAtten [10], respectively.

(3) Overwhelming complexity due to lack of cross-stage optimization. Based on online-softmax transformation, FA-2 employs a local tiling strategy to disaggregate the softmax into fragment operations. While the reduction in I/O complexity has been achieved, it comes with an intensive computational overhead, making it unsuitable for dynamic sparsity scenarios in LTPP. For example, when processing a sequence of length 1k with a tiling size of $B_c=4$, FA-2 must repeatedly perform exponentiation and comparison across blocks to en-

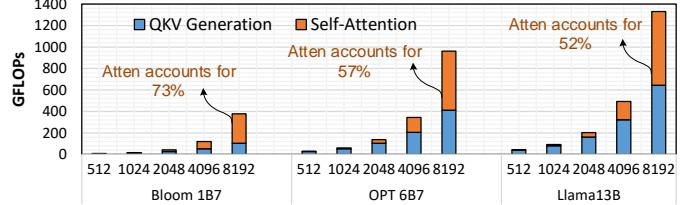


Fig. 7: Computational complexity between QKV and attention.

sure correctness of the global softmax result. This results in approximately $1.5 \times$ more computation compared to the ideal implementation, resulting in a $3.37 \times$ increase in computational energy consumption, due to the introduction of a significant number of expensive exponentiation operations.

In summary, the key bottleneck hindering DS accelerators in LTPP scenarios is the lack of cross-stage coordination, thus severely limiting memory and inference efficiency optimizations. Unfortunately, previous DS accelerators [5–11] all overlook this critical aspect. A³ [5], ELSA [6], DOTA [8] and Sanger [7] focus on alleviating the computing overhead of prediction stage, but all overlook the overhead introduced by memory accesses, let alone exploit the potential for cross-stage optimization. While SpAtten [10] and Energon [11] realize the challenge of off-chip memory access, SpAtten’s head pruning causes irreversible accuracy degradation, and Energon’s multi-round filter incurs prohibitive latency. Both therefore fail to handle the severe memory access overhead in LTPP scenarios. Overall, existing works focus on isolated, single-stage optimization and overlook cross-stage opportunities. This limitation restrains their ability to handle LTPP, which motivates our design of a cross-stage DS accelerator.

B. Extension for Spatial Architecture

Although the monolithic STAR accelerator effectively exploits attention sparsity to achieve high computational efficiency, its single-core architecture faces fundamental scalability limitations. As LLMs rapidly expanding in both parameter size and sequence length [16], the demands on on-chip memory and computational resources increasingly surpass the capabilities of a single core [17]. To overcome these challenges, multi-core spatial architectures are emerging as a promising solution for scalable LLM inference. Therefore, it is both significant and timely to investigate how the STAR accelerator can be effectively deployed within spatial architectures.

To this end, we leverage the STAR as the basic computing core, exploring the multi-core partial architecture. This spatial architecture enables parallel execution across cores, distributed storage of key/value matrices, and reduced off-chip memory traffic through localized data reuse. However, the irregular and dynamic nature of sparse attention poses new challenges in inter-core communication and cross-core data orchestration. These challenges motivate our study of spatial dataflow and communication algorithms tailored to STAR-based multi-core architecture, aiming to enable scalable and efficient attention processing for ultra-large LLM workloads.

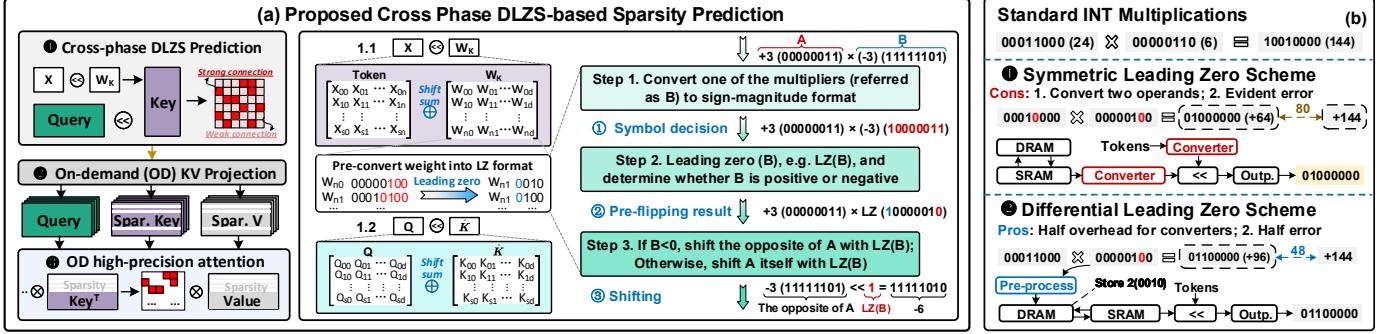


Fig. 8: (a) Cross-phase DLZS sparsity prediction and low-power PSP strategy in DLZS. (b) Comparison of the proposed DLZS with the traditional symmetric leading zero scheme (SLZS) used in [9].

IV. OPTIMIZATIONS FOR CROSS-STAGE TILING

Fig. 6 (a) summarizes three STAR optimizations addressing the challenges in Sect. III-A. First, in the pre-computation stage, we introduce a cross-phase DLZS prediction scheme that is a multiplication-free method for estimating attention sparsity, while also extending the sparsity to KV generation. Second, we propose SADS, a distributed sorting strategy that partitions a long segment into multiple sub-segments, reducing sorting complexity and enabling cross-stage tiling. Third, we design SU-FlashAttention, which utilizes top- k information to streamline the update operations in standard FlashAttention. Together, these techniques form STAR’s cross-stage tiling workflow, which alleviates off-chip memory access and end-to-end latency, as shown in Fig. 6 (b).

A. Cross-Phase DLZS Sparsity Prediction

Current DS works typically preserve dense QKV generation, leading to computation inefficiency. As shown in Fig. 7, QKV generation can dominate the total cost for short sequences—exceeding attention when Bloom1B7 and OPT6B7 have fewer than 2k and 4k tokens, respectively. In contrast, attention becomes the bottleneck only beyond these lengths. Therefore, it is essential to jointly optimize QKV generation and self-attention to enhance overall efficiency.

To this end, we propose a cross-phase, on-demand sparsity mechanism. Instead of blindly generating all KVs, STAR only generates those KVs, whose attention scores are within the top- k range, as illustrated in Fig. 8(a). However, it requires the *pre-compute* stage first estimate \hat{K} , based on which then speculates the \hat{A} . Unfortunately, even utilizing low-precision multiplication (e.g. half-precision with MSBs only), it still incurs prohibitive power overhead. Therefore, a power- and accuracy-efficient prediction paradigm is essential.

$$x = \text{Sign} \times M \times 2^{W-LZ}, \quad LZ \in \mathbb{Z}[1, \dots, W]. \quad (3)$$

Based on the log-domain transformation, we propose a multiplier-free paradigm, named *differential leading zero scheme (DLZS)*. *Differential* means: for a multiplication operation, only one operand undergoes a logarithmic-domain transformation to produce its leading-zero (LZ) format, in contrast to prior work [9] that applies logarithmic transformation to

both operands. The underlying mathematical principle is as follows. For an INT-type number x , it can be represented by the Eq. (3), where W is the quantized bitwidth, M stands for the mantissa lying within $[0, 1]$, and LZ denotes its leading-zero value. Based on the representation, the multiplication of two INT numbers can be rewritten as Eq. (4a). By approximating M_y as 1, we can derive the final computation formula as Eq. (4b). Since the bitwidth W is fixed for a given model, DLZS directly shifts x based on LZ_y to estimate their multiplication, without expensive multipliers.

$$x \cdot y = \text{XOR}(S_x, S_y) M_x \cdot 2^{(W-LZ_x)} M_y \cdot 2^{(W-LZ_y)} \quad (4a)$$

$$\approx \text{XOR}(S_x, S_y) M_x \cdot 2^{(W-LZ_x+W-LZ_y)} \quad (4b)$$

However, it is non-trivial to achieve the *differential* style log-domain computation. A key challenge is the bit flipping induced by sign determination after shifting, which results in prohibitive power overhead and may offset the benefits of log-domain approximation. To this end, we design a *preflipping via symbol prediction (PSP)* strategy, as shown in Fig. 8 (a) right. Initially, one operand B is converted to a sign-magnitude representation. The LZ count of B , i.e., $LZ(B)$, is then computed, followed by checking the sign of B . If B is negative, the opposite of the other operand (that is A) is shifted. Otherwise, the operand A itself is shifted. In this way, we can avoid the substantial power overhead from bit flipping.

As depicted in Fig. 8 (a), the cross-phase DLZS prediction leverages a potential opportunity to further reduce the predicted power consumption. Specifically, since the weights are pre-known and keep unchanged during inference, DLZS pre-converts W_k into LZ format. This allows the first phase, that is *Key prediction phase* (1.1), to directly shift the input based on the pre-converted $LZ(W_k)$, without any extra LZ coding. In the subsequent *Attention prediction phase* (1.2), to alleviate the inaccuracy accumulation, we apply LZ encoding for Q , instead of \hat{K} . Overall, compared to previous DS prediction strategies, the proposed cross-phase DLZS features lower conversion overhead and a wider range of sparsity acceleration.

As depicted in Fig. 8 (b), compared to the traditional symmetric leading zero scheme (SLZS) used in FACT [9], the proposed DLZS offers three key advantages: **a) Lower conversion overhead**. This is because DLZS performs the

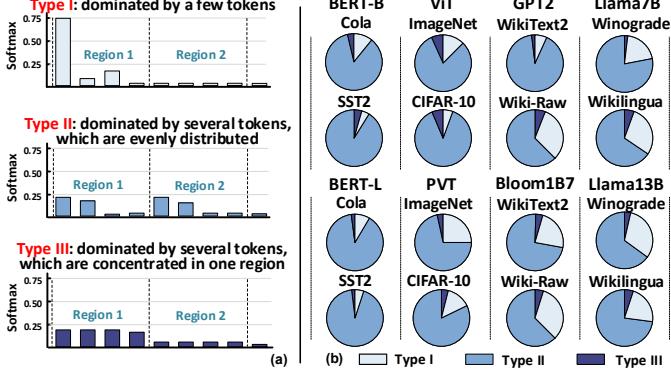


Fig. 9: The data distribution of attention across diverse LLMs.

leading-zero conversion on only one operand, while the SLZS needs to process both. **b) Higher accuracy.** This is because the error of the leading-zero scheme originates from the loss of information carried by the bits following the most significant ‘1’. Since DLZS performs the leading-zero conversion on only one multiplier, it introduces less error and achieves higher precision. **c) Reduced memory access.** The reduction in memory access stems from the fact that DLZS only needs to load a 4-bit LZ value that has been pre-converted offline, whereas SLZS must load the full 8-bit operand.

B. Sphere-search Aided Distributed Sorting (SADS)

To identify important tokens from the estimated attention \hat{A} , existing DS methods typically apply sorting or thresholding at the entire row level. However, these coarse-grained approaches incur significant computational overhead for long sequences, rendering them impractical for LTPP scenarios.

We begin by analyzing the softmax function and show that it naturally supports lightweight max-based decision-making. Without loss of generality, consider a two-element vector $[x_0, x_1]$ processed by softmax, where x_1 is the max element and $x_1 = x_0 + \Delta$. As shown in Eq.(5), the softmax output for x_0 decays exponentially with its distance Δ from x_1 . This observation implies that, after softmax, an element’s contribution is inversely correlated with its distance from the max: the farther it is, the smaller its impact on the final output.

$$x_1 = x_0 + \Delta \Rightarrow \text{softmax}(x_0) = \frac{e^{x_0}}{e^{x_0} + e^{x_0 + \Delta}} = \frac{1}{1 + e^{\Delta}} < \frac{1}{e^{\Delta}} \quad (5)$$

Since softmax exponentially amplifies the differences between elements, its output is mainly determined by the dominant tokens in the inputs. We classify the data distribution of each self-attention row into three types, as shown in Fig.9 (a). Type I: A few highly dominant tokens; Type II: Larger tokens evenly distributed across different regions; Type III: Larger tokens concentrated in a specific region. To characterize their distribution during inference, we analyze attention from several representative models and tasks, including BERT-B/L, ViT, PVT, GPT2, Bloom-1B7, Llama-7B, and Llama-13B. The analysis covers tasks spanning logic reasoning, text generation, and vision benchmarks (e.g., SST2, ImageNet, WikiText2, Winogrande), each based on 4k randomly sampled rows. Across all scenarios, Type II dominates with an average

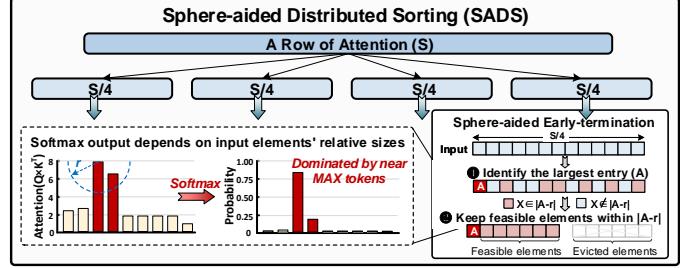


Fig. 10: Sphere-search aided distributed sorting (SADS).

proportion of about 73%. Meanwhile, Type I is more likely to occur in ViT, GPT, and LLaMA, with an average proportion of 22%, whereas in the encoder-based model BERT, it is only 12%. This discrepancy may be attributed to the local similarity features in images and the recent token characteristics inherent to auto-regressive decoder. By contrast, Type III occurs with negligible probability, which even approaches zero in both GPT-2 and LLaMA. This is likely due to the increased token length, where larger tokens tend to be distributed throughout the context, rather than being concentrated in a specific region.

For Type I and Type II, which together account for more than 95% of the total distribution, their primary commonality lies in their dispersed and relatively uniform attention distribution. This characteristic enables local maxima across different regions to be reasonably regarded as global maxima. Therefore, a strategy that can efficiently capture local maxima within Type I and Type II distributions in a cheap manner naturally represents a more effective token selection strategy.

To address this, we propose a *distributed sorting strategy named SADS*, which also incorporates a concept analogous to a spherical radius to enable early termination of redundant sorting for trivial data. As illustrated in Fig. 10, SADS first partitions an ultra-long sequence into n sub-segments (e.g., 4) for distributed sorting. Each segment then selects its own top- k/n largest values. Instead of naive sorting, SADS introduces a *spherical radius-based pruning criterion* for early termination. Specifically, it first identifies the largest entry A ① then defines a feasible region centered at A , denoted as $x \in |A - r|$. The subsequent top- k/n sorting is restricted to elements in the feasible region ②. For an element x , if its distance to the max satisfies $\Delta > r$, it is eliminated. This is justified by the analysis in Eq. (5), which shows the softmax value of an element x decreases exponentially with its distance Δ from the maximum. The radius r is user-configurable and is empirically set to 5 in our experiments. Under this setting, the softmax value of elements outside the feasible region drops below 0.0067, indicating their negligible contribution to the attention output.

Complexity Analysis. Compared with standard sorting of complexity $O(SSk)$, SADS reduces it to $O(SSk\rho/n)$, where k denotes the top- k ratio ($0 < k \leq 1$), n is the number of sub-segments, and ρ represents the remaining element ratio after early termination. In a typical setting with $S = 1024$, $n = 4$, $k = 0.25$, and $\rho = 0.4$ (with $r = 5$), the complexity of SADS is only about 10% of that of standard sorting. Notably, the specific number of sub-segments (e.g. tiling size) of each

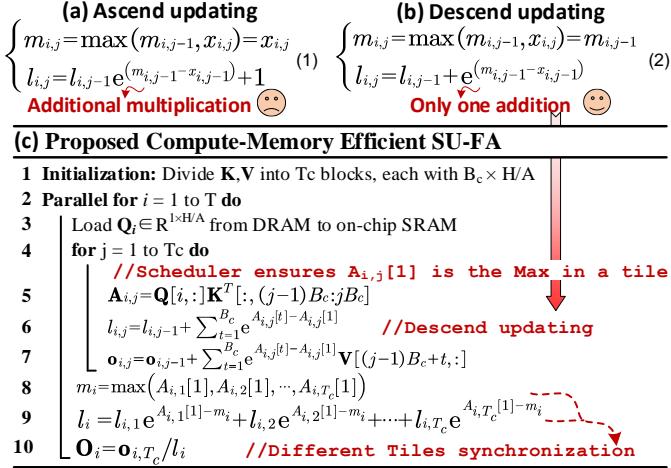


Fig. 11: (a) Redundant operations in FA. (b) Formulas for diverse updating orders. (c) Procedure of SU-FA.

layer is required to collaborate with SU-FA and is obtained by the DSE in Appendix A.

C. Sorted-Updating FlashAttention (SU-FA)

Existing DS methods alleviate the computational burden of attention but fail to optimize memory access, whereas FA reduces memory access at the cost of non-matmul overhead. To this end, we propose SU-FA, which utilizes the top- k information from DS to mitigate the max-value refreshes in FA, enhancing both computational and memory efficiency.

As shown in lines 5 and 7 of Fig. 5, redundant computations in FA mainly result from max-value updates across tiles and associated exponential operations. An intuitive approach to mitigate these inefficiencies is to leverage top- k information, which helps reduce these redundant computations. One straightforward scheme is to directly compute the max score based on the estimated index and then perform the attention calculation using the conventional softmax. However, there are two critical issues: 1) Due to the approximation property of DLZS, the estimated max index may be inaccurate, potentially leading to numerical overflow. 2) The separate computation of the max score incurs additional computational burden.

To address this, we propose a *successive sorted-updating* strategy. *Successive* means: it retains the same serial computation paradigm as FA, thus avoiding the overhead of separately computing the max score. *Sorted* indicates that the updating process is performed in accordance with the order of top- k indices. Naturally, there are two potential update orders: *ascending* and *descending*. This leads to the next question: which update order is more effective? Fig. 11 (b) presents the update formulas derived from top- k simplification for different strategies. Though both strategies eliminate the comparisons for the max value, they still lead to discrepancies in multiplication overhead. Compared to descend updating, ascend updating results in an additional multiplication operation at each step. While this discrepancy may seem trivial, in long-sequence scenarios, it would be accumulated and transformed into prohibitive overhead. For instance, with an 8k token

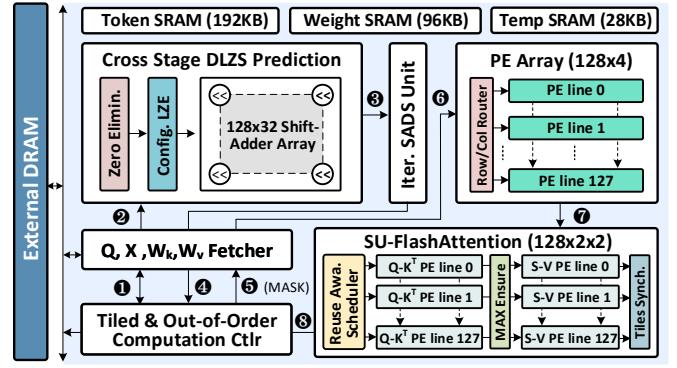


Fig. 12: High-level block diagram for the STAR accelerator.

sequence, ascend updating incurs 2.1×10^6 more multiplication operations, compared to descend updating. Therefore, SU-FA adopts the descend updating strategy by default.

To sum up, in the algorithm optimization of STAR, the granularity of sub-segment S_i and top- k jointly define an interesting design space. Smaller S_i leads to higher SU-FA overhead due to synchronization across sub-segments and exacerbates performance degradation due to more fragmented computation, but reduces sorting complexity. By contrast, larger S_i raises sorting overhead but lowers SU-FA overhead, and generally yields better accuracy. To this end, we present an efficient design space exploration (DSE) strategy to explore the optimal sub-segment size. Please refer to Appendix A.

V. ARCHITECTURE AND HARDWARE INNOVATION

§V-A introduces the overall STAR accelerator architecture, which effectively translates the theoretical algorithmic benefits into practical speedup. A detailed description of the primary hardware components is provided in Appendix B. Next, §V-B discusses the extension of STAR into the spatial architectures.

A. Overall Architecture

Fig. 12 depicts the high-level architecture of STAR accelerator, which consists of six main components: a cross-stage DLZS prediction unit, an SADS sorting unit, a PE array for KV on-demand generation, an SU-FA execution unit, a tiled & and out-of-order scheduler, and on-chip SRAM. STAR is designed to process 128 queries in parallel. First, the controller instructs the fetcher to load corresponding Q , X and W_k into the on-chip SRAM ①. Then, the DLZS unit begins cross-stage prediction, which generates an estimated attention scores \hat{A} ②. Next, based on the \hat{A} , the SADS unit identifies the indices of the corresponding key matrix for the top k/n scores in each row ③. The indices are then sent to the scheduler ④, which generates an intermediate binary mask ⑤ to guide the PE array in producing the necessary KVs ⑥. Once the KVs are generated on-demand ⑦, the SU-FA unit is triggered to execute the designed *descend updating* workflow, as presented in Sec. IV-C. The final results are stored back to off-chip DRAM ⑧.

B. Spatial Extension Design for STAR

Multi-chiplet integration offers a promising path to large-scale systems [18, 19]. Typically, it employs a 2D mesh

topology-based spatial architecture, as mesh exhibits lower wiring costs and superior scalability compared to alternative topologies [20] such as Fat-tree, Dragonfly and Switch-leaf.

As depicted in Fig. 13, we construct a 5×5 two-dimensional (2D) mesh spatial architecture, where each node instantiates the STAR accelerator as the fundamental computing unit. Notably, the choice of a 5×5 configuration represents a balanced trade-off between parallel scalability and communication overhead, while also following the design scale adopted in Dojo [21]. Each unit comprises a private SRAM block for local data storage and a dedicated router supporting five-directional routing (north, south, east, west, and local), thereby enabling flexible and low-latency communication across the mesh. The computing units are interconnected via a 2D mesh Network-on-Chip (NoC), enabling efficient horizontal and vertical data transfers for parallel execution and dynamic workload balancing. DRAM modules are symmetrically placed on both sides of the mesh. Memory requests issued by compute units are transmitted over the NoC using dynamic routing, and the requested data is returned to the corresponding units via multi-hop paths.

However, achieving efficient attention parallelism on the 2D mesh spatial architecture is non-trivial and presents two primary challenges: First, at the dataflow level, it is essential to carefully orchestrate the communication paths of Q, K, and V to minimize global traffic while maintaining regular and predictable communication patterns. Second, at the communication-logic level, it must be meticulously designed to ensure that the logical data movement aligns precisely with the 2D mesh physical topology, for mitigating tail latency.

To address these challenges, in this section, we first introduce a dedicated distributed attention dataflow strategy termed *DRAAttention*. Then, to enable its effective realization on the 2D mesh physical topology, we further propose an optimized logical communication algorithm, referred to as *MRCA*.

1) Distributed Ring-flow-based Attention (DRAAttention):

Fig. 14 illustrates the data partitioning and computation scheduling strategy of DRAAttention on the 5×5 2D mesh architecture. Before computation, both the Query tensor $\in \mathbb{R}^{S \times d_h}$ and the Input tensor $X \in \mathbb{R}^{S \times H}$ are partitioned only along the sequence dimension, while the hidden dimension remains intact. The Query tensor is divided into 25 sub-blocks of size $\in \mathbb{R}^{S/25 \times d_h}$, which are distributed across both the rows and columns of the STAR array. Specifically, consecutive groups of five sub-blocks are mapped to the five STAR units within the same row, with each unit processing one sub-block. By contrast, the input tensor X is divided into only five sub-blocks of size $\mathbb{R}^{S/5 \times H}$, which are mapped along the column dimension, with each sub-block shared by the five STAR units within the same column.

Each STAR unit on-demand generates the required KV

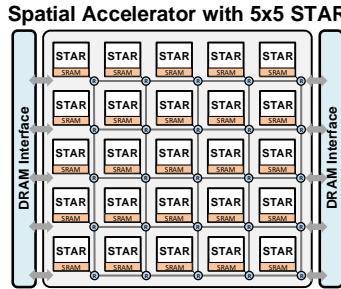


Fig. 13: 5×5 spatial archi. setup.

tensors corresponding to its current sub-block X_i . Once the computation begins, the STAR unit executes the sparse attention as depicted in Section V-A. During this process, for any STAR unit, while computing attention between its local Q sub-block and the generated KV tensors, the STAR unit concurrently transmits its Q sub-block to the next STAR unit and receives a Q sub-blocks from the preceding unit, as depicted in the Fig. 14 (b). If the computation time exceeds the time required for transferring the Q sub-blocks, no additional communication overhead is incurred. Further, certain intermediate data, such as the local maximum m_i and partial sum l_i , are propagated alongside the Q sub-blocks. These values are incrementally updated at each time step to support normalization and maintain numerical stability. In this way, after the fifth time step, each STAR unit updates the attention result of its own Q sub-block using the global maximum value and generates the corresponding final attention results.

Overall, DRAAttention offers two significant advantages. First, its Query-driven communication leads to low global communication overhead, as the Query tensor is substantially smaller than the K/V tensors. This reduction in communication volume helps alleviate NoC congestion. Second, DRAAttention enables substantial overlap between computation and communication, effectively mitigating the potential performance degradation caused by exposed communication latency and improving compute unit utilization.

However, from the perspective of communication logic, DRAAttention relies on a near-circular (a.k.a ring-style) data transfer pattern during execution, as illustrated by the red arrow in Fig. 14(b). This logical communication behavior requires physical wrap-around links to enable cross-boundary data propagation. However, in practical hardware implementations, such a wrap-around link is generally infeasible due to physical constraints such as excessive routing distance and signal integrity issues. Therefore, a 2D mesh topology cannot, in practice, provide the degree of interconnectivity required to directly realize this communication pattern.

To resolve the mismatch between logical communication requirements and physical topology constraints, we propose an optimized communication scheduling algorithm, named *MRCA*. This approach enables DRAAttention to achieve the desired circular communication behavior within a standard mesh topology, without incurring additional physical interconnect

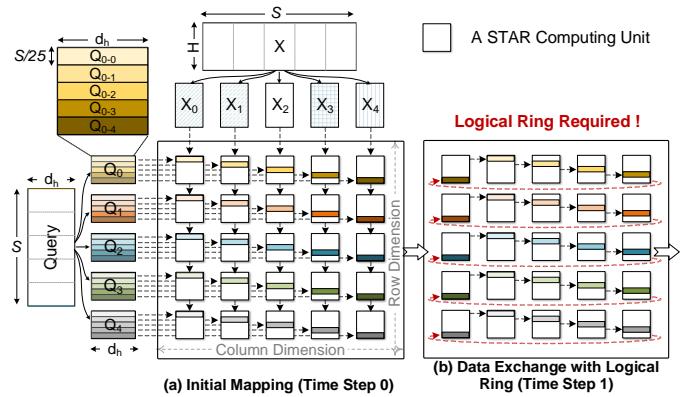


Fig. 14: Tailored dataflow for DTAttention mechanism.

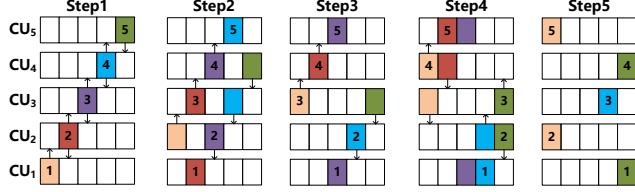


Fig. 15: Q stream in a 1×5 spatial accelerator. CU_i denotes the computing unit with ID i . Different colors represent different data chunks. Blank chunks indicate that the corresponding chunk does not reside on the CU. The number i indicates the chunks involved in the CU_i 's computation at the step.

costs, thereby facilitating efficient attention parallelism.

2) Mesh friendly Ring Communication Algorithm (MRCA):

Instead of forcing a logical ring topology onto a physical mesh topology, MRCA leverages the natural characteristics of the mesh to construct a congestion-free orchestration that is logically equivalent to a ring. The central innovation is inspired by the behavior of reflux tides, analogous to the reverse wave of a progressing wave. This mechanism, referred to as *Reflux*, enables elegant transfer of the Q-stream and scales effectively to any number of computing units. The details of MRCA is summarized in Alg. 1, while an execution example on a 1×5 spatial architecture (i.e., a 1D mesh with five computing units, CUs) is depicted in Fig. 15. For illustrative purposes, the 1D mesh is drawn in a vertical layout.

In Alg. 1, the communication mechanism is categorized into two types of operations: progress wave (lines 4-9) and reflux tides (lines 10-19). Progress wave occurs in nearly all N (5) steps. The transferred Query sub-blocks (For short, we denote them as chunks) behave like waves spreading from the center outwards. We consider the transfer from a CU with a smaller ID to one with a larger ID as an upward wave (lines 4-6 in Alg. 1), and the transfer from a CU with a larger ID to one with a smaller ID as a downward wave (lines 7-9).

As illustrated in Step2 of Fig. 15, following lines 4 and 7 in Alg. 1, CU_2 , CU_3 , and CU_4 participate in both upward and downward wave transfers. Taking CU_2 as an example, in Step2, it stores chunk1 and chunk3. It forwards chunk1 to CU_3 (lines 5-6 in Alg. 1) and simultaneously passes chunk3 downwards to CU_1 (lines 8-9). Therefore, the progress wave achieves the transmission of each chunk to the dies on both sides through the spread of chunks from the center, ensuring that each CU does not store more than 2 chunks per step.

The reflux tides occur after the $\lceil \frac{N}{2} \rceil$ -th step, with the primary aim of ensuring that each CU can complete computations on different chunks within N steps. To maintain computational balance, each CU would store at most two chunks in certain steps but would only compute one of them. For instance, consider CU_2 in Fig. 15, which stores chunk1 and chunk3 at Step 2; CU_2 only utilizes chunk3 for computation in this step, neglecting chunk1. Without reflux tides, where chunk1 resides solely on CU_2 at Step 2 due to progress waves, CU_2 would be unable to compute the output related to chunk1.

Hence, the introduction of reflux tides ensures that chunks required by each CU return to them at specific steps, enabling complete output computation. Taking CU_3 as an example,

Algorithm 1: MRCA Communication Algorithm

```

Input:  $N$  CUs connected with 1-D mesh topology
for  $t = 1$  to  $N$  do
    parallel for  $n \in CUs$ 
         $src \leftarrow n;$ 
        if  $t \leq src < N$  then
             $dest \leftarrow src + 1, i \leftarrow src - t + 1;$ 
             $send(src, dest, chunk[i]);$ 
        if  $1 < src \leq N - t + 1$  then
             $dest \leftarrow src - 1, i \leftarrow src + t - 1;$ 
             $send(src, dest, chunk[i]);$ 
        if  $t > \lfloor N/2 \rfloor$  then
            if  $t = \lfloor N/2 \rfloor + 1$  then
                CUs replicate original chunks locally
            else
                if  $t - \lfloor N/2 \rfloor \leq src < t$  then
                     $dest \leftarrow src + 1,$ 
                     $i \leftarrow src + N - t + 1;$ 
                     $send(src, dest, chunk[i]);$ 
                if  $N - t + 1 < src \leq N - t + 1 + \lfloor N/2 \rfloor$ 
                then
                     $dest \leftarrow src - 1,$ 
                     $i \leftarrow src - N + t - 1;$ 
                     $send(src, dest, chunk[i]);$ 

```

Output: Data orchestration in logical ring on mesh

which corresponds to line 11 in Alg. 1, at Step 3, all CUs transfer chunks while locally replicating them. Consequently, even though other dies do not transfer chunks to CU_3 in Step 3, CU_3 retains both chunk1 and chunk5. Following line 14 in Alg. 1, at Step 4, CU_3 participates in reflux tides communication. According to lines 15-16 in Alg. 1, during Step 4, CU_3 transfers chunk1 to CU_2 and chunk5 to CU_4 , thereby reintroducing chunk1 and chunk5 to CU_2 and CU_4 , respectively. This operation compensates for the missed computations involving chunk1 and chunk5 on CU_2 and CU_4 during Step 2. Therefore, reflux tides not only ensure computational completeness but also minimize communication costs and storage overhead. In this way, MRCA ensures the implementation of DTAttention's dataflow within a mesh physical topology.

VI. EVALUATION

A. Experimental Setup

We evaluate the performance of STAR on several representative Transformer models and tasks using the NVIDIA A100 GPU. For computer vision (CV) tasks, we use the latest PVT model [22] for ImageNet-1k classification, by fine-tuning from the checkpoint pre-trained on ImageNet-21k. For NLP tasks, we select the BERT-base and BERT-large models [3], which are evaluated across eight tasks from the GLUE benchmark and SQuAD v1.1. Furthermore, we assess GPT-2 [1], Bloom-1.7B [23], and Llama7B/13B [24] on language modeling tasks including WikiText-2, WikiLingua, Wiki-raw, and Winogrande. For each task, fine-tuning is performed on the NVIDIA A100 GPU following token pruning to recover accuracy. We evaluate models using task-specific metrics: F1 score for SQuAD and STS-B, accuracy for MRPC, RTE, SST-2, QNLI and Winogrande, ROUGE-1 for WikiLingua, and

TABLE II: Accuracy of Diverse Transformer Models with FP16, INT16 and STAR Configurations (S: Standard, A: Aggressive).

Model	BERT-Base						BERT-Large						GPT-2	PVT	Bloom1B3	LLaMA-7B	LLaMA-13B	
Task [‡]	MRPC	RTE	SST2	STS2	SQuAD	QNLI	MRPC	RTE	SST2	STS2	SQuAD	QNLI	Wiki2.	ImageN.	WikiLi.	WikiRaw	Wiki2.	Winog.
FP16	85.3%	67.1%	91.9%	83.9	87.4%	80.8%	90.0%	70.1%	94.8%	86.5	93.2%	90.7%	18.3	83.6%	44.3	24.3	10.2	70.1%
INT16	85.3%	67.1%	91.9%	83.7	87.3%	80.8%	90.0%	70.0%	94.8%	86.5	93.2%	90.7%	18.7	83.4%	44.1	24.5	10.5	70.1%
STAR (S)	85.3%	67.0%	91.9%	83.7	87.3%	80.6%	89.9%	70.0%	94.8%	86.4	93.1%	90.7%	18.7	83.4%	44.1	24.5	10.5	70.1%
STAR (A)	83.9%	66.7%	91.8%	83.2	87.1%	80.1%	88.5%	69.2%	93.2%	85.1	91.4%	90.6%	18.9	81.9%	43.8	25.3	10.8	69.4%

[‡] MRPC, RTE, SST-2, QNLI, ImageNet, Winogrande are evaluated by accuracy. SQuAD and STS-B are evaluated by F1-score and Pearson Correlation, respectively. WikiLingua is evaluated by ROUGE-1. WikiText2 and WikiText Raw are evaluated by perplexity, where lower is better. Standard: 0% drop vs. INT16; Aggressive: $\leq 2\%$ drop vs. INT16.

perplexity for Wikitext-2 and Wiki-raw, Top-1 accuracy for ImageNet. For evaluation, we adopt the dense INT16 model as the baseline. We progressively increase the pruning ratio in steps corresponding to a 0.2% accuracy loss (Eq.(6)) and record the resulting sparse configurations, including the layer-wise top- k settings and the segment sizes. In this way, we obtain a spectrum of sparsity configurations under different controlled levels of accuracy loss. We evaluate two configurations: standard (0% loss), aggressive (2% loss), representing the minimal and maximal performance optimizations.

$$Loss = \Delta\mathcal{M} = \frac{|\mathcal{M}_{\text{baseline}} - \mathcal{M}_{\text{STAR}}|}{\mathcal{M}_{\text{baseline}}} \times 100\%. \quad (6)$$

Overall Accuracy. All models are sourced from Pytorch and HuggingFace. The INT16 baselines are derived via post-training quantization. As shown in Table. II, the INT16 baseline incurs less than a 1% average accuracy drop from FP16. For reasoning tasks such as MRPC, QNLI and Winogrande, the accuracy degradation caused by INT16 quantization is negligible, typically below 0.5%. This aligns with prior work [25], which suggests that classification and reasoning tasks, due to their discrete output space and robustness to quantization noise, exhibit a high tolerance for low precision.

For hardware evaluation, we conduct the RTL design for the STAR accelerator and utilized Synopsys DC with TSMC 28nm CMOS technology, to estimate the area and power consumption of the logic parts. On-chip SRAM bandwidth and power are evaluated with CACTI. For off-chip DRAM, we model memory behaviors with Ramulator [26] and apply the methodology from [27] to estimate I/O power consumption. We then simulate the RTL with Verilator to extract the actual cycles for each stage, and use this data to build a cycle-level simulator for evaluating end-to-end performance.

For GPU comparisons, we run benchmarks on the A100 platform with the SOTA TensorRT-LLM framework. Execution time is measured by inserting `torch.cuda.synchronize` at run boundaries. For power measurement, we first record the system's idle power with `nvidia-smi`, then repeatedly execute workloads to obtain the total power. The dynamic power is derived by subtracting idle power from the total power.

For evaluation of the spatial architecture, we use ASTRA-sim [28], an open-source simulator for distributed machine learning systems, integrated with STAR's cycle-accurate timing model. The simulator is reconfigured with the corresponding NoC communication and DRAM access model, enabling throughput simulation under various workloads.

B. Algorithm Performance

Experimental Settings. For the objective function of DSE (see Appendix A), the coefficient α controls the weight of

the top- k sorting cost, while β governs the weight of the SU-FA's exponential cost. We first conduct numerical experiments to define appropriate search ranges for each hyperparameter. During retraining, we apply grid search with successive halving to efficiently identify optimal parameters for each model. Depending on the input-token length and parameter size for different models, α/β is set as 0.24/0.31 (BERT-B/L), 0.2/0.24 (ViT), 0.4/0.42 (GPT-2), 0.53/0.56 (Bloom-1.7B), and 0.58/0.63 (Llama-7B/13B), respectively. We then conduct a search for 200 iterations using each learning rate (1e-1, 5e-2, 1e-3) to identify the optimal tiling configuration.

Top- k Hit. Fig. 17 (a) profiles the hit rate between the predicted top- k by SLZS+SADS, DLZS+SADS and the true top- k during GPT-2 inference. As shown, SLZS+SADS exhibits relatively low hit rates, remaining below 93% for the top-20% scenario, largely attributable to estimation errors introduced by leading-zero conversion of both operands. In contrast, DLZS+SADS consistently achieves higher hit rates. For the top-20% setting, the hit rate exceeds 97%, with degradation below 90% observed only in shallow layers under the stricter top-5% setting. As the network depth increases, the hit rate rises above 95%, owing to deeper layers extracting higher-level features that yield more distinguishable attention scores. Notably, the higher hit rate in deeper layers helps preserve inference accuracy, as numerical precision near output layers has a greater impact on final performance.

Fig. 17 (b) further shows that DLZS+SADS incurs at most a 4.8% accuracy drop without fine-tuning compared to the dense baseline, which is reduced to within 2% with appropriate fine-tuning. In summary, since DLZS+SADS achieves a high hit rate in deeper layers, which have the most direct impact on the final output, it can effectively preserve the final model performance on downstream tasks. Moreover, with additional fine-tuning, the accuracy loss can be further minimized, making it adaptable to scenarios with different precision requirements.

TakeAway1. *In top- k settings, decreasing k increases pruning, which helps to reduce computational complexity but may compromise accuracy. STAR provides designers with the flexibility to adaptively balance accuracy and acceleration by adjusting the pruning ratio according to practical needs.*

Ablation Study. We first set up an ablation study to evaluate the low-complexity benefits of DLZS, SADS, and SU-FA by comparing them with the baseline scheme. The baseline is assumed to employ 4-bit multiplication during the *pre-compute stage*, vanilla sorting in *top- k stage*, and traditional FA during the *formal-compute stage*. For fairness, we kept the loss of different models within 2%. As depicted in Fig. 18, compared to the baseline, DLZS achieves an average 18% complexity reduction, which is mainly benefits from its multiplier-free design and the use of only half the number of

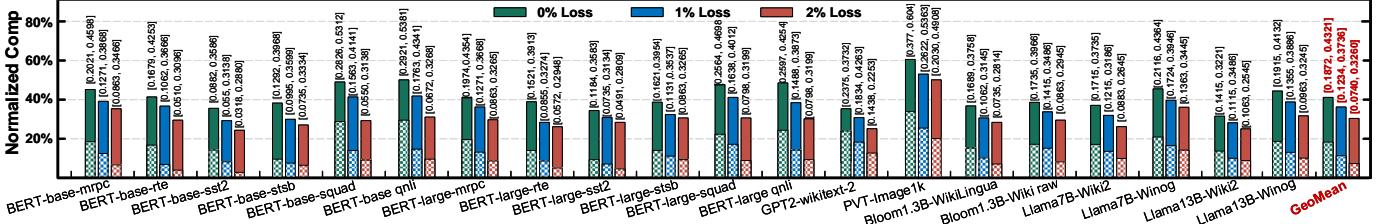


Fig. 16: Comp reduction by LP with varied loss tolerance. [X,Y] is normalized Comp in Atten and Atten+QKV, respectively.

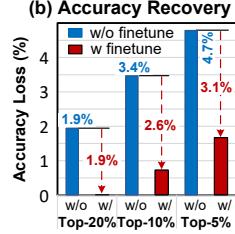
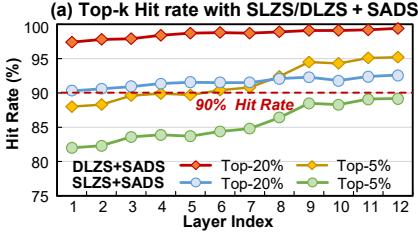


Fig. 17: (a) Layer-wise top- k hit rates of SLZS/DLZS and (b) End-to-end accuracy analysis of DLZS on GPT-2 WikiText-2.

converters. In addition, SADS and SU-FA together contribute an additional 10% reduction, through distributed segmented sorting and the elimination of redundant non-linear operations using top- k hints. Overall, compared to traditional schemes, STAR’s software optimizations achieves a 28% reduction in computational complexity for the same token sparsity, thus enabling it to efficiently handle LTPP scenarios.

Accuracy & Reduced Complexity Trade-off. Fig. 18 (b) shows the impact of top- k ratio γ on accuracy and reduced complexity (RC) using ViT on ImageNet (CV) and Llama7B on Winogrande (reasoning). Overall, a smaller γ results in more aggressive pruning, which may decrease accuracy but increase reduced complexity. There are some key observations: For CV tasks (ImageNet), accuracy drops noticeably when $\gamma < 0.2$. In contrast, for text reasoning tasks (Winogrande), the model is more tolerant to pruning, with accuracy degrading evidently only when $\gamma < 0.15$. This may be because text tasks rely on vital tokens for inference, resulting in higher redundancy. On the other hand, the gains in reduced complexity start to diminish when $\gamma < 0.15$. This is because overly aggressive pruning harms crucial tokens and limits further complexity reduction. Therefore, to strike a balanced trade-off between accuracy and complexity, we empirically set the top- k ratio preferentially within the range of 0.15–0.2.

Overall Complexity Reduction. Fig. 16 depicts the computation reduction achieved by STAR’s sparsity prediction mechanism (LP) across various tasks, using dense models as the baseline and allowing 0%, 1%, and 2% accuracy loss. As observed, Text-based sentiment tasks (e.g., STT-2 and STSB) exhibit high sparsity, enabling over 90% computation reduction with 1% loss. This is because typically, only a few key terms suffice to capture the sentiment polarity. In contrast, image tasks generally contain more critical information, with lower data redundancy compared to text classification datasets, leading to reduced sparsity. As a result, the computation reduction is limited to around 73% under a 1% loss. Overall,

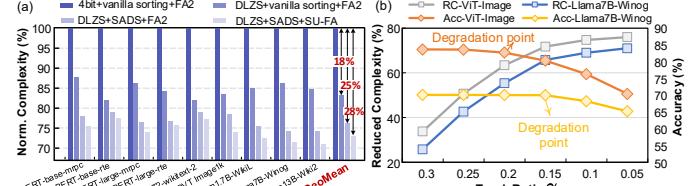


Fig. 18: (a) Complexity reduction for DLZS, SADS, SU-FA. (b) Trade-off between accuracy and reduced complexity (RC).

LP achieves significant computation reduction. On average, with accuracy losses of 0%, 1%, and 2%, the LP decreases the attention + QKV computation by 56.8%, 62.6%, and 67.4%, respectively. When focusing on the attention part, computation reduction increases to 81.3%, 87.7%, and 92.6%, respectively.

TakeAway2: *Textual tasks typically exhibit higher sparsity than vision tasks. Consequently, STAR’s sparsity prediction achieves extremely high computation reduction (over 90%) in high-redundancy scenarios such as text classification.*

C. Architecture Evaluation

Throughput over GPU. Fig. 19 shows the throughput gain of STAR over the Nvidia A100 GPU across different tasks. As observed, naively applying the LP mechanism on the GPU results in a limited throughput gain of about $1.08 \times$ – $1.78 \times$. This is because the GPU’s coarse-grained parallelism cannot handle token-level fine-grained sparsity, nor can it manage cross-stage on-demand generation and the fine-grained token reallocation of SU-FA. By contrast, the STAR accelerator, with its customized datapath, efficiently pipelines the cross-phase DLZS, distributed SADS, and synergistically updating SU-FA modules, thus achieving a sparse utilization almost triple higher than GPUs. On average, STAR achieves speedups of $6.3 \times$ / $7.0 \times$ / $9.2 \times$ with 0%/1%/2% accuracy loss, respectively. Fig. 22 (a) illustrates STAR’s effectiveness in reducing memory accesses. Compared to the baseline using vanilla dynamic sparsity, STAR with RASS achieves an average memory access reduction of 23%. With the incorporation of SU-FA and tiled dataflow, the reduction increases to 79%.

Throughput gain breakdown. Fig. 20 shows the breakdown of throughput gain, with the baseline being the execution of the dense model on the A100 GPU. With the dedicated ASIC datapath, STAR is $1.5 \times$ faster than the GPU baseline. LP is then applied to find trivial tokens, reducing computation by $3.1 \times$. However, the performance only improves by $1.15 \times$. This is because dynamic pruning requires pre-computation and

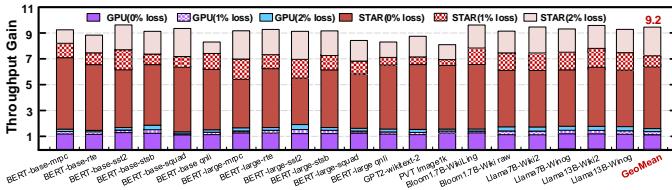


Fig. 19: Throughput gain of STAR over LP on A100 GPU.

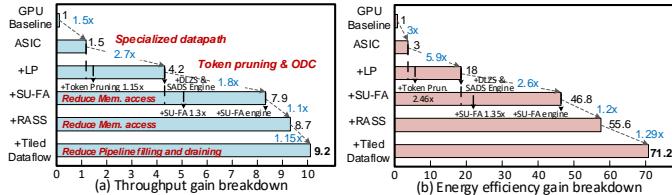


Fig. 20: Throughput and energy efficiency gain breakdown.

top- k to identify redundant tokens, which becomes a bottleneck without dedicated hardware. With the tailored DLZS and SADS engines, performance further improves by $2.7\times$. Similarly, directly applying SU-FA yields only a $1.3\times$ gain due to Max value errors often causing circuit stalls. In contrast, the tailored SU-FA engine can achieve a $1.8\times$ gain. Also, RASS and tiled dataflow together contribute around a $1.27\times$ speedup.

Area, Power and Energy: Fig. 21 presents the area and power breakdown of the STAR accelerator implemented in TSMC 28nm technology. It has a total area of 5.69 mm^2 and a power consumption of 949.85 mW . The LP part (DLZS + SADS) occupies only 18.1% of the area and consumes 14.1% of the power, yet it delivers a $2.8\times$ throughput gain. Fig. 22 (b) further depicts the energy efficiency improvement over the A100 GPU. Under 0%/1%/2% accuracy degradation, STAR realizes $49.8\times/51.6\times/71.2\times$ efficiency gain, respectively. Fig. 20 (b) also shows the efficiency gain breakdown. The DLZS and SADS engines provide efficiency gains of $2.58\times$ and $2.3\times$, respectively, while the SU-FA and RASS units together contribute a $3.12\times$ improvement.

Fig. 23(a) shows the throughput performance under a single-core architecture when adopting the LP+SU-FA+RASS+Tiled dataflow design compared with the baseline across different SRAM capacities. The baseline corresponds to the conventional attention computation paradigm without softmax tiling. The results indicate that with 316 kB of SRAM, the LP+SU-FA+RASS+Tiled dataflow design achieves saturated throughput, while the baseline remains memory-bound due to frequent

TABLE III: Summary and comparison with SOTA works.

	FACT	Energon	ELSA	STAR
Acceleration for [†]	C only	C only	C only	C & M
Optimization level*	S. Stg	S. Stg	S. Stg	C. Stg
Tech.[nm] & Freq [Hz]	28/500M	45/1G	40/1G	28/1G
Area[mm ²]	6.03	4.20	1.26	5.69
Power[W]	0.22	2.72	1.5	3.45
Throughput[GOPS]	928	1153	1090	24423
Energy Effi.[GOPS/W] [‡]	2754	450	1004	7183
Area Effi.[GOPS/mm ²] [‡]	154	709	1765	4292

[†]C: Computation. M: Memory access; * S. Stg: Single stage; C. Stg: Cross stage; [‡]Scaled to the 28nm technology and 1.0V with $f \propto s$, power (core) $\propto (1/s)(1.0/Vdd)^2$ as [27, 29], where $s = \text{Tech}/28\text{nm}$.

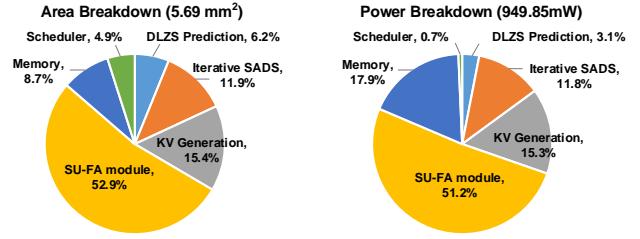


Fig. 21: Area and power breakdown of STAR accelerator.

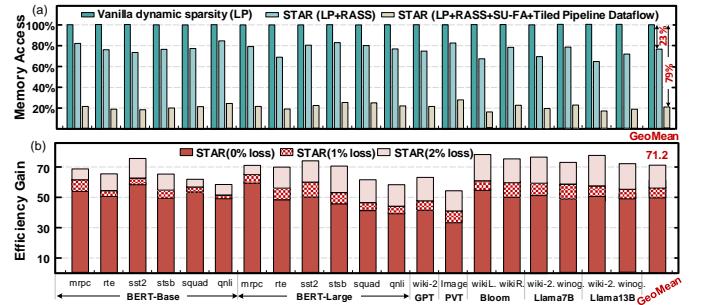


Fig. 22: (a) Memory access reduction of STAR. (b) Energy efficiency gain of STAR accelerator over Nvidia A100 GPU.

memory accesses and thus fails to reach peak throughput.

Fig. 23(b) illustrates the performance gains achieved under a 5×5 spatial architecture when applying the DRAttention+MRCA optimization scheme. The baseline is configured without these optimizations as well as SU-FA and RASS. In the spatial architecture, memory constraints become more pronounced due to the increased competition for memory bandwidth among multiple compute units. With 412 kB of SRAM, the baseline achieves only 3 TOPS, whereas DRAttention+MRCA reaches 24.1 TOPS— $12\times$ higher—thanks to its efficient dataflow and enhanced data reuse opportunities.

TakeAway3: *Without customized dataflow and communication algorithms (e.g., DRAttention) to mitigate memory-access overhead, employing a multi-core architecture can actually degrade throughput due to contention for memory resources.*

D. Comparison with Existing Accelerators

Sanger [7], FACT [9], Energon [11], SpAtten [10] and ELSA [6] are SOTA Attention DS accelerators. However, they all prioritize computation optimization and overlook memory access, which becomes the dominant bottleneck as computation efficiency improves. While SpAtten reduces memory traffic via head pruning and Energon considers compute–memory balance, both suffer from task and model dependency, lacking of generality. Overall, prior designs fail to jointly optimize computation and memory access under sparsity.

In contrast, STAR employs a holistic FlashAttention-like scheme that tiles all DS stages and exploits sorting information for cross-stage optimization. Table III compares the characteristics and metrics of FACT, Energon, ELSA and STAR. STAR’s energy efficiency (with cross-stage compute and memory optimizations) reaches 7183 GOPS/W, which is $2.6\times$, $15.9\times$ and $7.2\times$ (tech normalized [27, 29]) higher than FACT, Energon and ELSA (single-stage computation-only

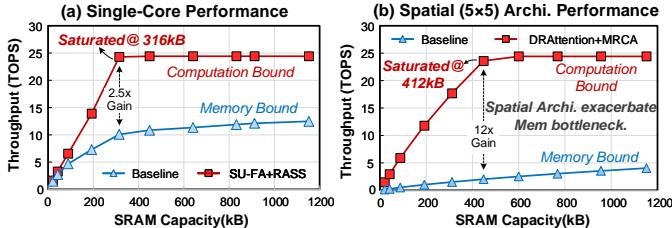


Fig. 23: Impact of SRAM size on average single-core throughput with/without memory-access optimizations under (a) single-core with 256GB/s DRAM bandwidth (b) a multi-core configuration with 512GB/s total (shared) DRAM bandwidth, i.e., 20.5 GB/s effective bandwidth per core.

optimization). This improvement results from the fine-grained data flow enabled by collaborative cross-stage optimization, effectively reducing off-chip memory accesses. Additionally, STAR achieves 4292 GOPS/mm² in area efficiency, which is 27.1 \times , 6.1 \times and 2.4 \times greater than FACT, Energon and ELSA, respectively. The area efficiency gain is primarily due to algorithm-hardware co-optimization for low complexity.

E. Performance of Spatial Architecture

We first conduct an ablation study to evaluate the effectiveness of the *DRAAttention* dataflow and the customized *MRCA* communication algorithm in improving STAR’s performance on spatial architectures. Evaluation configurations are summarized in Table IV. The baseline adopts a Ring-Attention (ICLR’23) [30], where KV tensors are transmitted among compute units following a ring topology, without any topology-aware or communication-efficiency optimizations.

As shown in Fig. 24 (a), under the 5 \times 5 scenario, *DRAAttention* achieves an average 3.1 \times throughput gain, primarily due to its strategy of transmitting smaller Query tensors. This allows communication latency to be better overlapped with computation, thereby improving compute unit utilization. However, this logic-level optimization is inherently constrained by the mismatch between logical and physical topologies. With the integration of *MRCA*, throughput further increases by 3.6 \times , attributed to efficient communication orchestration that mitigates tail latency from boundary compute units. As the system scales to a 6 \times 6 mesh (Fig. 24 (b)), the gain from *DRAAttention* diminishes to 3 \times , whereas *MRCA* achieves a higher gain of 4.2 \times . This is because the larger-scale system results in more limited average memory bandwidth, emphasizing *MRCA*’s superior capability in reducing contention and tail latency.

Fig. 24 (c) compares the performance of diverse spatial architectures integrated with diverse computing units. The baseline is assumed to adopt the same NVDLA compute

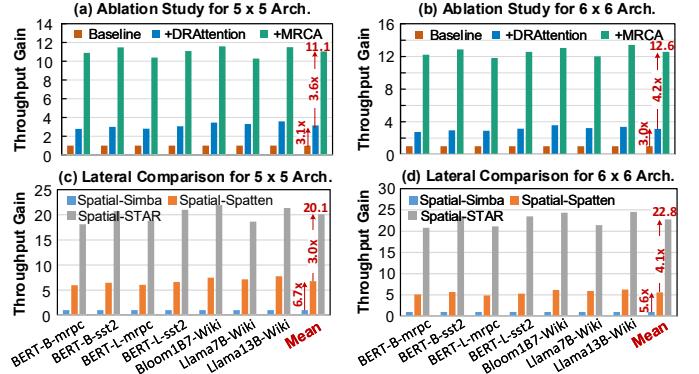


Fig. 24: (a)(b) Ablation studies for DRAAttention and MRCA. (c)(d) Lateral comparison for diverse computing units.

architecture as Simba [31], utilizing SIMD-based multiple parallel vector MAC units to perform dense attention computations, referred to as *Spatial-Simba*. In contrast, the spatial architectures using Spatten [10] and STAR as the compute unit are referred to as *Spatial-Spatten* and *Spatial-STAR*, respectively. As depicted in Fig. 24 (c), under 5 \times 5 scenario, *Spatial-Spatten* archives an average 6.7 \times gain while *Spatial-STAR* improves this gain further to 20.1 \times . For *Spatial-Spatten*, its benefits mainly stem from the computation acceleration for token sparsity. Unfortunately, the heavy IO burden still hinders its performance gain, especially for the spatial architecture scenarios, where memory access heavily relies on the transfer by NoC routers. Heavy traffic increases the contention in the NoC network, which in turn degrades the throughput. In contrast, strategic cross-stage tiling in *Spatial-STAR* efficiently alleviates the IO burden, making it well-suited for spatial architecture. When scaling up to a 6 \times 6 architecture, as depicted in Fig. 24 (d), the performance gap caused by the I/O optimization capability becomes further amplified. The average throughput gain of *Spatial-Spatten* decreases to 5.6 \times , whereas that of *Spatial-STAR* further increases to 22.8 \times .

Scalability. STAR handles long sequences by combining spatial unfolding (DRAAttention) and temporal unfolding (SU-FA, Tiling), enabling scalability to arbitrarily long inputs. For ultra-long sequences, it only requires assigning additional workload to each STAR unit, which can process them by extending the number of time steps.

Load-Balance Handling. The DRAAttention dataflow inherently mitigates load imbalance through uniform workload partitioning and overlapped compute–communication scheduling. 1) Each STAR unit receives an equal-sized query sub-block and performs local attention while simultaneously transmitting and receiving sub-blocks along the mesh. This pipelined execution hides minor timing variations among units, ensuring near-uniform utilization. 2) Further, the MRCA employs a reflux phase that synchronizes boundary tiles and compensates for straggling nodes, effectively reducing tail latency. These mechanisms together maintain balanced progress even under small variations in compute or link latency.

TABLE IV: Spatial archi. configuration parameters.

Modules	Parameters	Configurations
Logic Part	Logic Die Area	324mm ²
	Die Router	Input-queued Architecture
	Routing Algorithm	Dimension-Order
	Die-to-Die Interconnect	250GB/s, 20ns, 1.0pJ/bit
DRAM Part (HBM 2)	HBM-die Area / Data Rate	240 mm ² / @2Gbps/pin
	Access Bandwidth	512GB/s, 100ns, 6.0pJ/bit

VII. RELATED WORKS AND DISCUSSION

Efficient Transformer Accelerators. Numerous works [5–11, 32–37] aim to improve Transformer inference efficiency. However, most of these efforts focus on alleviating attention’s quadratic computation, by static sparsity [32–34], dynamic sparsity [5–11], and hybrid sparsity [36]. Unfortunately, in many cases, memory access is the de facto bottleneck in both power and end-to-end latency, particularly in LTPP scenarios. To address this, STAR jointly optimizes computation and memory access, offering substantial improvements over prior works. STAR adopts a cross-stage holistic optimization strategy. This unique methodology enables cross-stage tiling, facilitating fine-grained dataflow tiling that accelerates inference while significantly reducing off-chip memory access.

Neural Network Accelerators with Sparsity. A variety of ASIC and FPGA accelerators leverage sparsity to optimize neural network inference performance [38–41]. However, most existing work focuses on pre-trained static sparse weights, while STAR utilizes LP to predict dynamic sparsity on-the-fly, particularly exploiting the argmax approximation of softmax, which requires active detection. This makes traditional static zero-based sparsity methods ineffective. While some works have targeted activation sparsity [42] and both weight and activation sparsity [43], they are all based on near-zero sparsity, failing to address the top- k sparsity that STAR targets.

VIII. CONCLUSION

This paper presents STAR, a scalable Transformer accelerator for LTPP, enabled by a cost-efficient log-domain predictor, distributed sorting and simplified FlashAttention. To further scale across spatial architectures, we propose a customized dataflow and a tailored communication algorithm. Evaluated on 5×5 and 6×6 spatial architectures, STAR delivers 20.1× and 22.8× throughput gains over the baseline.

REFERENCES

- [1] A. Radford, J. Wu, R. Child *et al.*, “Language models are unsupervised multitask learners,” *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.
- [2] J. Achiam, S. Adler, S. Agarwal *et al.*, “GPT-4 technical report,” *arXiv:2303.08774*, 2023.
- [3] J. Devlin, M.-W. Chang *et al.*, “BERT: Pre-training of deep bidirectional Transformers for language understanding,” *arXiv:1810.04805*, 2018.
- [4] M. AI, “The LLaMA 4 herd: The beginning of a new era of natively multimodal AI innovation,” <https://ai.meta.com/blog/llama-4-multimodal-intelligence/>, 2025.
- [5] T. J. Ham, S. J. Jung, S. Kim *et al.*, “A³: Accelerating attention mechanisms in neural networks with approximation,” in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2020, pp. 328–341.
- [6] T. J. Ham, Y. Lee, S. H. Seo *et al.*, “ELSA: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks,” in *Proc. IEEE Annu. Int. Symp. Comput. Archit.*, 2021, pp. 692–705.
- [7] L. Lu, Y. Jin, H. Bi *et al.*, “Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture,” in *Proc. IEEE Int. Symp. Microarch.*, 2021, pp. 977–991.
- [8] Z. Qu, L. Liu, F. Tu *et al.*, “DOTA: Detect and omit weak attentions for scalable Transformer acceleration,” in *Proc. ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2022, pp. 14–26.
- [9] Y. Qin, Y. Wang, D. Deng *et al.*, “FACT: FFN-attention co-optimized Transformer architecture with eager correlation prediction,” in *Proc. IEEE Int. Symp. Comput. Archit.*, 2023, pp. 1–14.
- [10] 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.
- [11] Z. Zhou, J. Liu *et al.*, “Energon: Toward efficient acceleration of Transformers using dynamic sparse attention,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 1, pp. 136–149, 2022.
- [12] T. Dao, D. Fu, S. Ermon *et al.*, “FlashAttention: Fast and memory-efficient exact attention with IO-awareness,” *Adv. Neural Inf. Process. Syst.*, vol. 35, pp. 16344–16359, 2022.
- [13] S. Williams *et al.*, “Roofline: An insightful performance model for multicore architectures,” *Commun. ACM*, vol. 52, no. 4, pp. 65–76, 2009.
- [14] T. Dao, “FlashAttention-2: Faster attention with better parallelism and work partitioning,” *arXiv:2307.08691*, 2023.
- [15] R. P. Brent and P. Zimmermann, *Modern computer arithmetic*. Cambridge Univ. Press, 2010, vol. 18.
- [16] J. Kaplan, S. McCandlish *et al.*, “Scaling laws for neural language models,” *arXiv:2001.08361*, 2020.
- [17] Y. Hu, X. Lin, H. Wang *et al.*, “Wafer-scale computing: advancements, challenges, and future perspectives [Feature],” *IEEE Circuits Syst. Mag.*, vol. 24, no. 1, pp. 52–81, 2024.
- [18] H. Wang, Z. Wang, H. Wang *et al.*, “WATOS: Efficient LLM training strategies and architecture co-exploration for wafer-scale chip,” *arXiv preprint arXiv:2512.12279*, 2025.
- [19] H. Wang, T. Wei, Z. Wang *et al.*, “TEMP: A memory efficient physical-aware tensor partition-mapping Framework on wafer-scale chips,” *arXiv preprint arXiv:2512.14256*, 2025.
- [20] K. Kandalla *et al.*, “Designing topology-aware collective communication algorithms for large scale infiniband clusters,” in *IEEE Int. Symp. Parallel Distrib. Process., Workshops PhD Forum*, 2010, pp. 1–8.
- [21] E. Talpes, D. Williams *et al.*, “Dojo: The microarchitecture of Tesla’s exa-scale computer,” in *Proc. IEEE Hot Chips 34 Symp.*, 2022, pp. 1–28.
- [22] W. Wang, E. Xie, X. Li *et al.*, “Pyramid vision Transformer: A versatile backbone for dense prediction without convolutions,” in *Proc. IEEE Int. Conf. Comput. Vision*, 2021, pp. 568–578.
- [23] T. Le Scao, A. Fan, C. Akiki *et al.*, “Bloom: A 176B-parameter open-access multilingual language model,” *arXiv:2211.05100*, 2022.
- [24] H. Touvron, T. Lavril, G. Izacard *et al.*, “LLaMa: Open and efficient foundation language models,” *arXiv:2302.13971*, 2023.
- [25] B. Jacob, S. Kligys, B. Chen *et al.*, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2704–2713.
- [26] Y. Kim *et al.*, “Ramulator: A fast and extensible DRAM simulator,” *IEEE Comput. Archit. Lett.*, vol. 15, no. 1, pp. 45–49, 2015.
- [27] H. Wang, J. Fang, X. Tang *et al.*, “SOFA: A compute-memory optimized sparsity accelerator via cross-stage coordinated tiling,” in *Proc. IEEE/ACM Int. Symp. Microarch. (MICRO)*, 2024, pp. 1247–1263.
- [28] W. Won, T. Heo *et al.*, “Astra-sim2. 0: Modeling hierarchical networks and disaggregated systems for large-model training at scale,” in *IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, 2023, pp. 283–294.
- [29] H. Wang, B. Cheng, X. Tan *et al.*, “An efficient approximate expectation propagation detector with block-diagonal neumann-series,” *IEEE Trans. Circuits Syst. I*, vol. 70, no. 3, pp. 1403–1416, 2023.
- [30] H. Liu, M. Zaharia, and P. Abbeel, “Ring attention with blockwise transformers for near-infinite context,” *arXiv:2310.01889*, 2023.
- [31] Y. S. Shao, J. Clemons, R. Venkatesan *et al.*, “Simba: Scaling deep-learning inference with multi-chip-module-based architecture,” in *Proc. IEEE/ACM Int. Symp. Microarch.*, 2019, pp. 14–27.
- [32] H. You, Z. Sun, H. Shi *et al.*, “ViTCoD: Vision Transformer acceleration via dedicated algorithm and accelerator co-design,” in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, 2023, pp. 273–286.
- [33] B. Li, S. Pandey, H. Fang *et al.*, “FTRANS: Energy-efficient acceleration of Transformers using FPGA,” in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Design*, 2020, pp. 175–180.
- [34] G. Shen, J. Zhao, Q. Chen *et al.*, “SALO: An efficient spatial accelerator enabling hybrid sparse attention mechanisms for long sequences,” in *Proc. ACM/IEEE Des. Autom. Conf.*, 2022, pp. 571–576.
- [35] H. Wang, H. Wang, Z. Wang *et al.*, “PADE: A predictor-free sparse attention accelerator via unified execution and stage fusion,” *arXiv preprint arXiv:2512.14322*, 2025.
- [36] J. Zhao, P. Zeng, G. Shen, Q. Chen, and M. Guo, “Hardware-software co-design enabling static and dynamic sparse attention mechanisms,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2024.
- [37] H. Wang, H. Wang, Z. Yue *et al.*, “BETA: A Bit-Grained Transformer Attention Accelerator With Efficient Early Termination,” *IEEE Trans. Circuits Syst. II*, 2025.
- [38] R. Hojabr, A. Sedaghati, A. Sharifian *et al.*, “SPAGHETTI: Streaming accelerators for highly sparse GEMM on FPGAs,” in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, 2021, pp. 84–96.
- [39] B. Asgari, R. Hadidi, T. Krishna *et al.*, “ALRESCHA: A lightweight reconfigurable sparse-computation accelerator,” in *Proc. IEEE Int. Symp.*

- High-Perform. Comput. Archit.*, 2020, pp. 249–260.
- [40] H. Wang, Z. Zhang, X. You, and C. Zhang, “Low-complexity winograd convolution architecture based on stochastic computing,” in *Proc. IEEE Int. Conf. Digit. Signal Process.*, 2018, pp. 1–5.
 - [41] H. Wang, Z. Wang, Z. Yue *et al.*, “MCBP: A Memory-Compute Efficient LLM Inference Accelerator Leveraging Bit-Slice-enabled Sparsity and Repetitiveness,” *arXiv preprint arXiv:2509.10372*, 2025.
 - [42] J.-W. Jang, S. Lee, D. Kim *et al.*, “Sparsity-aware and re-configurable NPU architecture for Samsung flagship mobile SoC,” in *Proc. IEEE Int. Symp. Comput. Archit.*, 2021, pp. 15–28.
 - [43] Y. N. Wu, P.-A. Tsai, S. Muralidharan *et al.*, “HighLight: Efficient and flexible DNN acceleration with hierarchical structured sparsity,” in *Proc. IEEE Int. Symp. Microarch.*, 2023, pp. 1106–1120.