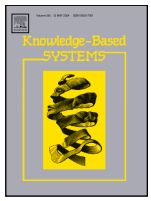
Knowledge-Based Systems 333 (2026) 115018

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys



Accelerating long-context inference of large language models via dynamic attention load balancing

Jie Ou [a,1], Jinyu Guo [a,1], Shuaihong Jiang [a], Xu Li [b], Ruini Xue [c,*], Wenhong Tian [a], Rajkumar Buyya [d]

[a] School of Information and Software Engineering, University of Electronic Science and Technology of China No.4, North Jianshe Road, Chengdu, 610054, Sichuan, China
[b] School of Computer Science and Technology, Tongji University No. 4800 Cao'an Road, Shanghai, 201804, China
[c] School of Computer Science and Engineering, University of Electronic Science and Technology of China No.2006, Xiyuan Ave, Chengdu, 611731, Sichuan, China
[d] Quantum Cloud Computing and Distributed Systems (qCLOUDS) Lab, School of Computing and Information Systems, The University of Melbourne Grattan Street, Parkville, 3010, Victoria, Australia

ARTICLE INFO

Keywords:
Large language models
Long-context inference
Attention load balancing
Sparse attention
Efficiency

ABSTRACT

Large language models (LLMs) have demonstrated exceptional performance across various natural language processing tasks. However, their quadratic complexity of attention mechanisms results in inefficiency during long-context inference. Although existing research has employed sparse attention techniques to enhance LLM efficiency in long-context scenarios, we observe that these methods introduce heterogeneous attention computation patterns across different heads, leading to GPU load imbalance and resource idling during practical deployments. To address this challenge, we propose FlexAttn, a novel inference framework that dynamically generates attention load balancing strategies tailored to input context lengths. Our framework enhances resource utilization during long-context prefilling by scheduling attention heads within each layer according to the searched strategies. Specifically, FlexAttn first conducts head-level profiling to collect computational characteristics and then searches for a load balancing strategy based on the current context length and profiling data. To minimize runtime overhead, we partition and reorganize the weights before inference execution. Furthermore, as the computational overhead is considerably larger than the I/O overhead in long-context inference, we employ a cross-prefetch strategy for each transformer layer to enhance efficiency. Extensive experiments demonstrate that when applied to state-of-the-art long-context techniques, our framework achieves a throughput improvement of 34.95 % to 40.9 % on LLaMA3-8B across context lengths ranging from 160k to 768k tokens. Notably, our proposed approach remains orthogonal to conventional model parallelism and sparse attention techniques, enabling complementary performance enhancements when integrated with existing accelerating methods.

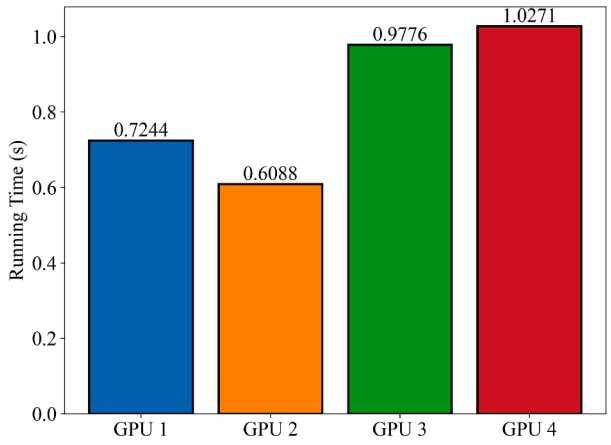
1. Introduction

Recent years have witnessed significant advancements in large language models (LLMs) based on decoder-only architectures [1–4], demonstrating exceptional performance across diverse natural language processing tasks [5–8]. This capability underscores their potential as a pathway toward artificial general intelligence. Growing adoption of LLM-powered applications has intensified demands for enhanced capacity to process extended text sequences. LLMs are expected to effectively handle extensive contextual information in various practical scenarios requiring codebase-level analysis [9,10], long-document question answering (QA) [11], in-context learning [12], and multiturn dialogue interactions [13], based on their long-context reasoning capabilities.

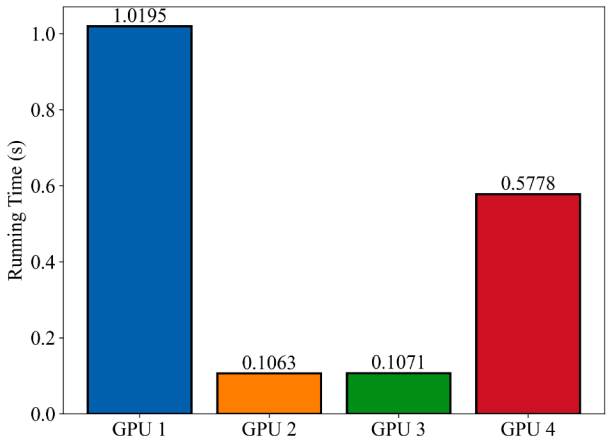
However, despite these advancements, critical challenges continue to persist in the practical deployment of LLMs, particularly regarding computational efficiency [14]. The quadratic time complexity inherent in conventional attention mechanisms poses substantial computational challenges while processing long contexts. Therefore, addressing this efficiency bottleneck through the systematic exploration of optimization strategies in long-context inference tasks constitutes a critical challenge for broadening the practical applicability of LLMs across diverse domains.

* Corresponding author.
E-mail addresses: 202211090813@std.uestc.edu.cn (J. Ou), guojinyu@uestc.edu.cn (J. Guo), jiang_shuaihong@std.uestc.edu.cn (S. Jiang), lx2024@tongji.edu.cn (X. Li), xueruini@uestc.edu.cn (R. Xue), tian_wenhong@uestc.edu.cn (W. Tian), rbuyya@unimelb.edu.au (R. Buyya).
[1] The two authors contribute equally to this work.

https://doi.org/10.1016/j.knosys.2025.115018
Received 7 March 2025; Received in revised form 25 September 2025; Accepted 28 November 2025
Available online 2 December 2025



(a) 4-way GPU parallel running time of layer 15 based on MInference with context length 160k.



(b) 4-way GPU parallel running time of layer 15 of DuoAttention with context length 160k.

Fig. 1. Comparison of running costs for the LLaMA-8B model on different GPUs with different sparse attention methods.

Current methodologies for improving the efficiency of large language models (LLMs) primarily follow two research approaches. The first approach focuses on enhancing computational resource utilization via parallel computing techniques. Researchers have developed multiple parallelization strategies including tensor parallelism [15], pipeline parallelism [16], data parallelism [17,18], and sequence parallelism [19–21]. Advanced frameworks such as DeepSpeed [22] facilitate flexible integration of these parallelization strategies. To simplify distributed training implementation, automation tools, such as AMP [23] and Alpa [24] employ automatic strategy searching algorithms that generate optimized hybrid configurations based on model architectures and hardware specifications. The second approach involves algorithmic optimizations that leverage sparsity patterns in attention computation. Recent investigations, including MInference [25], DuoAttention [26], RazorAttention [27], and RHME [28] have demonstrated that attention mechanisms can be decomposed into retrieval-oriented and non-retrieval components. Furthermore, sparsity optimization has been extended to broader architectural designs: SeqBoat [29] dynamically reduces sequence modeling overhead through sparse modular activation, SLTrain [30] improves pretraining efficiency by combining low-rank and sparse matrix decomposition, MsDL [31] enhances time series classification using multi-scale sparse recurrent structures. Through the systematic reduction in computational intensity in non-retrieval attention heads, these methods have substantially improved their efficiency in computational complexity and memory usage. Their empirical results indicate that proper parameter selection in such optimized architectures can match or even exceed the performance of standard LLMs while significantly reducing resource demands.

In long-context techniques where attention heads are categorized and the original computational paradigm is modified, thereby transforming parallel computation within individual layers from homogeneous to heterogeneous patterns during LLM inference. However, current parallelization approaches struggle to comply with heterogeneous computing demands arising from differentiated attention head processing in multihead attention mechanisms. We analyzed the runtime characteristics for the 15th layer of the LLaMA3-8B model[1] Fig. 1a and b under 4-GPU tensor parallelism configurations, implementing attention head classification strategies from MInference and DuoAttention. Our experimental results demonstrate substantial load imbalance across GPUs, with performance differentials reaching 43.77 % and 97.51 % between the slowest and fastest devices, respectively. This imbalance directly leads to increased end-to-end (E2E) latency and suboptimal GPU utilization efficiency. Fig. 2 shows that computational cost discrepancies among different attention patterns can escalate exponentially escalate with context length expansion. As existing studies focus on long-context applications and new sparse attention algorithms continue to emerge, the development of parallelization techniques that can effectively address the load imbalance in sparse attention mechanisms becomes imperative. Therefore, achieving such advancements is crucial for harnessing the full potential of sparse attention approaches and eventually enhancing the performance and operational efficiency of LLMs in long-context reasoning tasks.

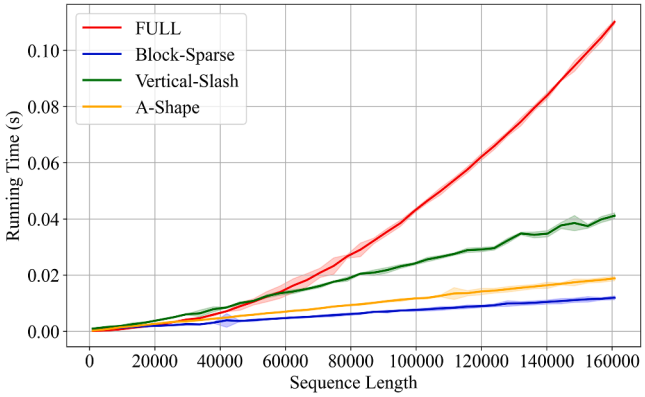


Fig. 2. Time costs of different head patterns at different sequence lengths.

To address the abovementioned challenge, we present the automatic attention equalization for long-context inference (*FlexAttn*) framework, which dynamically optimizes attention load balancing across transformer layers according to input sequence length, thereby enhancing the computational efficiency of LLMs for long-context processing. Specifically, our *FlexAttn* comprises four distinct phases: basic data collection, strategy searching, weight partition & reorganization, and cross-prefetch inference pipeline. Through these coordinated phases, we achieve balanced computational resource utilization for LLMs in long-context inference scenarios, thereby accelerating LLM inference efficiency. In the first phase (basic data collection), we systematically collect performance metrics across varying context lengths using predefined configurations: a specified sparse attention mechanism, target LLM architecture, and

[1] https://huggingface.co/gradientai/Llama-3-8B-Instruct-Gradient-1048k

hardware setup. This collected dataset serves as the foundation for subsequent strategy searches. To identify attention equalization strategies, we develop an attention load balancing solver (ALBS), which optimizes attention load balancing. Aiming at further enhancing the practicality of ALBS, we implement a cost modeling approach that eliminates the need for real hardware interaction during the search process, significantly reducing computational overhead. Following strategy identification, we address potential implementation challenges through our weight partition & reorganization (WPR) technique. This method strategically partitions attention layer weights according to individual head allocations derived from ALBS solutions and then reorganizes these matrix segments based on GPU grouping configurations. Preprocessing weight matrices in this manner helps eliminate runtime matrix indexing/slicing operations and optimize CPU-to-GPU data transfer efficiency. Finally, we implement a cross-prefetch inference pipeline (CPIP) designed for long-context processing characteristics. This memory-optimized solution maintains the weight of only one transformer layer in GPU memory at any execution stage while ensuring continuous computational resource utilization through strategic prefetching mechanisms. Consequently, more memory can be allocated for storing computational variables, enabling the processing of longer sequences under fixed resource constraints, thereby enhancing the efficiency of LLMs.

We apply our *FlexAttn* framework to state-of-the-art (SOTA) long-context inference methods, including MInference and DuoAttention, and we conduct extensive evaluations across various LLM architectures. Experimental results demonstrate that our framework achieves significant efficiency improvements, with throughput increases of 34.95 % to 40.9 % for LLaMA3-8B across context lengths ranging from 160k to 768k tokens. Notably, our approach remains orthogonal to conventional model parallelism techniques and existing sparse attention implementations, allowing for complementary performance gains when used in conjunction with these methods. Our primary advancements comprise four key aspects:

1. We propose *FlexAttn*, a novel framework that effectively addresses the challenge of computational load imbalance in long-context LLM inference, significantly improving GPU utilization efficiency for heterogeneous attention computations.
2. We design an attention load balancing solver that searches for balanced attention head combinations across different GPUs. By incorporating a cost model, ALBS eliminates the need for extensive hardware profiling during strategy searching, making it both efficient and practical.
3. We propose the weight partition & reorganization technique that strategically restructures attention layer weights, eliminating weight matrix indexing and partition during runtime and enhancing the efficiency. Furthermore, we develop a cross-prefetch inference pipeline that maintains the weight of only one transformer layer in GPU memory while ensuring continuous computational resource utilization through prefetching mechanisms.
4. We extensively evaluate *FlexAttn* across various LLM architectures and state-of-the-art long-context inference methods, including MInference and DuoAttention. Experimental results demonstrate significant performance improvements while maintaining compatibility with existing parallelization techniques and sparse attention methods.

The rest of this paper is organized as follows. Section 2 discusses existing research related to LLM inference acceleration. Section 3 introduces fundamental concepts of the transformer architecture. Section 4 details the architecture of *FlexAttn* and its four key components. Section 5 experimentally validates the efficiency of *FlexAttn*. Section 6 provides an extended experimental analysis with some intuitive understanding. Section 7 provides a comprehensive conclusion to the whole paper. Section 8 summarizes the limitations of this study and outlines future research directions.

Table 1
Summary of some representative works.

Category	Insights	Representation
Efficient Context Computation	Sparse Attention	[25,26]
	Key Value Eviction	[32,33]
	Text Compression	[34,35]
	Kernel-based Approximations	[36,37]
	State-space Models	[38,39]
	Retrieval	[40]
Parallelization Strategies	DP, TP, PP, SP	[15–17,20]
	Hybrid Parallel	[23,40]
Offline Inference Systems	Throughput Oriented	[22,41]

2. Related work

In recent years, remarkable advancements have been made in LLMs in terms of ultra-long context inference and parallelization acceleration. This section presents some existing studies from three perspectives: (1) long-context inference techniques primarily address the computational complexity of attention mechanisms and model scalability in extended sequence scenarios; (2) parallelization strategies focus on efficient GPU resource allocation using techniques including data parallelism, model parallelism, and sequence parallelism; and (3) offline inference systems emphasize throughput optimization. The subsequent Table 1 presents a concise taxonomy of relevant literature, while detailed analyses of the approaches listed are provided in the following subsections.

2.1. Efficient context computation

The computational complexity of attention layers presents significant challenges for LLMs processing extended sequences [42]. Various approaches have been proposed to enhance attention efficiency, including the sparse attention patterns [43–47], kernel-based approximations [36,37], and linear-complexity state-space models that replace traditional attention mechanisms [38,39]. Although effective, these solutions typically require architectural modifications and model retraining. Alternative methods attempt to reduce computation via pruning of redundant key-value (KV) vectors [32,48]. These methods have limitations in extending the context windows of LLMs without additional training, owing to distributional mismatch challenges caused by unseen positions.

Although recent token eviction methods have achieved significant speedup during decoding without requiring training [32,33,48–53], the computational overhead of the prefill phase has become substantial with the increasing prevalence of long-context application scenarios. To address this, methods such as InfLLM [40] reduce computational overhead via progressive prefilling. At each processing step, it strategically retrieves and utilizes cached KV pairs from historical states, implementing a dynamic sparsity mechanism to enhance the computation efficiency. Prompt compression techniques [34,35] address prefilling overhead via text compression. In addition, MInference [25] and DuoAttention [26] have emerged to accelerate long-text prefill processing based on the inherent sparse pattern of the attention, which is an inherent characteristic of LLMs. These approaches leverage the identified sparse computation patterns inherent in LLMs, achieving acceleration via predefined sparse attention patterns across different attention heads.

Existing solutions have demonstrated significant improvements in computational and memory efficiency through algorithmic and architectural enhancements. However, the application of these techniques might be further enhanced to achieve better runtime efficiency in practical deployments. This study primarily focuses on resolving the multi-GPU load imbalance issue caused by sparse attention techniques in the prefill phase (exemplified by MInference and DuoAttention), aiming to enhance the practical deployment efficiency of such acceleration methods. The proposed approach is orthogonal to and compatible with existing

optimization methods, enabling further efficiency gains when combined with these methods.

2.2. Parallelization strategies

Data-parallel [17,18] processing partitions computational workloads along the data dimension, distributing input batches across multiple devices to alleviate individual device load and enhance overall throughput. This approach requires each device to maintain complete model replication. By contrast, model parallelism addresses GPU memory limitations via parameter distribution across devices, primarily implemented using two methodologies: tensor parallelism [15] for horizontal layer parameter division across GPUs, and pipeline parallelism [16] for vertical layer-wise model partitioning. Even when single-GPU memory capacity suffices (e.g., through CPU offloading [54]), computational intensity may still prolong training durations. Advanced frameworks such as DeepSpeed [22] integrate three-dimensional (3D) parallelism, combining tensor, pipeline, and data parallelism to optimize parameters and optimizer state management under memory constraints. Complementary techniques, such as ZeRO-1 [55] and the distributed optimizer of Megatron [15] further optimize memory usage through fragmented optimizer state allocation across data-parallel workers. Sequence parallelism [19–21] splits the activation values in the sequence dimension. Compared with pipeline parallelism at the layer level, it is a more fine-grained pipeline parallel method, which improves the utilization of computing resources. The strategic combination of sequence and tensor parallelism enhances efficiency for large-scale model training. Consequently, modern LLM training systems typically integrate 3D parallelism with distributed optimizers and sequence parallelism.

Despite progress in training optimization, current parallelization strategies suffer critical deficiencies during inference operations, particularly in long-sequence contexts. While tensor parallelism mitigates parameter storage demands, computational heterogeneity among attention heads in differentiated attention mechanisms compromises parallelization effectiveness. Furthermore, existing approaches prioritize general computational optimization while inadequately addressing distinctive prefilling phase characteristics in long-context inference. Our *FlexAttn* alleviates these limitations via heterogeneous attention load balancing, systematically mitigating computational bottlenecks during prefilling to achieve substantial performance improvements in long-context inference tasks.

2.3. Offline inference systems

In addition to interactive applications such as chatbots, LLMs serve background tasks such as benchmarking [56], information extraction [57], data wrangling [58], and form processing [59]. These batch-oriented tasks prioritize throughput over latency. Frameworks such as FlexGen [41]and DeepSpeed [22] support such workloads across single/multi-GPU configurations through zig-zag execution patterns that maximize token throughput.

However, these frameworks primarily schedule the model weights and are designed for scenarios with shorter sequence lengths (e.g., 4K tokens). Conversely, our *FlexAttn* focuses on enhancing inference efficiency for extremely long texts (e.g., 512K tokens) by optimizing the load balancing of attention computations across multiple devices, building upon the foundation of long-sequence sparse attention algorithms.

3. Preliminaries

In this section, we first revisit the fundamental concepts of the transformer to lay the groundwork for understanding our proposed method. Contemporary LLMs predominantly employ the decoder-only transformer architecture. The core component of these architectures is the self-attention mechanism, which enables context-aware sequence processing.

Given an input sequence $(\langle s\rangle, x_1, x_2, \dots, x_{L-1})$ where $\langle s\rangle$ denotes the start token, the language model F estimates conditional probability distributions as follows:

$$P(x_t|x_1, x_2, \dots, x_{t-1}) = F(\langle s\rangle, x_1, x_2, \dots, x_{t-1}) \tag{1}$$

The model architecture consists of stacked N transformer layers: $F = f_N \circ f_{N-1} \circ \dots \circ f_1$, where the initial layer f_1 processes embedded input sequences and subsequent layers f_i $(i > 1)$ process outputs from the previous layer. Each transformer layer f contains two principal components: multihead self-attention (MHA) and feed-forward network (FFN). Our study specifically focuses on the GPU workload imbalance issues in MHA layers during sparse attention processing of long sequences.

The MHA mechanism performs position-aware contextual modeling through learnable projections. For input matrix $\mathbf{X} \in \mathbb{R}^{L\times d}$, L denotes the length of context and d denotes the hidden size of LLM. It generates queries ($\mathbf{Q}$), keys ($\mathbf{K}$), and values ($\mathbf{V}$) as follows:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V \tag{2}$$

where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d\times(h\times d_k)}$, h denotes the number of heads, and d_k is the size of each head. The attention computation proceeds as follows:

$$\mathrm{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V} \tag{3}$$

The FFN component employs two linear transformations with nonlinear activation:

$$\mathrm{FFN}(\mathbf{X}) = \sigma(\mathbf{X}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2 \tag{4}$$

where $\mathbf{W}_1 \in \mathbb{R}^{d\times d_{\mathrm{ff}}}$, $\mathbf{W}_2 \in \mathbb{R}^{d_{\mathrm{ff}}\times d}$, and σ denotes activation functions such as ReLU or GELU. Layer computations integrate residual connections as follows:

$$\tilde{\mathbf{X}} = \mathbf{X} + \mathrm{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \tag{5}$$

$$\mathbf{Y} = \tilde{\mathbf{X}} + \mathrm{FFN}(\tilde{\mathbf{X}}) \tag{6}$$

These connections mitigate gradient vanishing while enhancing information flow through the network. The quadratic computational complexity of self-attention presents significant challenges for long-text processing. Note that in many current models, such as LLaMA, FFN is referred to as a multi-layer perceptron (MLP), and we will use this terminology hereafter.

4. Methods

4.1. FlexAttn overview

Fig. 3 shows the multistage workflow of the proposed offline long-context inference system, which sequentially operates through four interconnected phases. First, the basic data collection phase establishes foundational configurations by specifying the target large language model (LLM), attention pattern techniques (e.g., MInference and DuoAttention), and hardware environment. This phase subsequently generates the attention head patterns report (AHPR) through layer-wise head-type analysis, followed by executing benchmark tests across varying sequence lengths on target devices to produce the runtime profiling report (RPR) containing the execution cost (Fig. 2). Next, the strategy searching phase employs our attention load balancing solver to process the AHPR and RPR data, deriving an attention-aware load balancing strategy adapted to specific input sequence lengths. Subsequently, the weight partition & reorganization phase restructures LLM weights according to attention-aware load balancing strategy specifications, replacing runtime indexing with preorganized matrix groups (e.g., merging $head_1$ and $head_5$, isolating $head_2$, and combining $head_3$ with $head_4$ (Fig. 3)) to optimize memory access patterns. Finally, the cross-prefetch inference pipeline phase processes the inputs through computation-prefetch interleaving, where weight loading for subsequent layers overlaps with current-layer computations, effectively hiding memory latency and maximizing GPU utilization throughout the accelerated inference process.

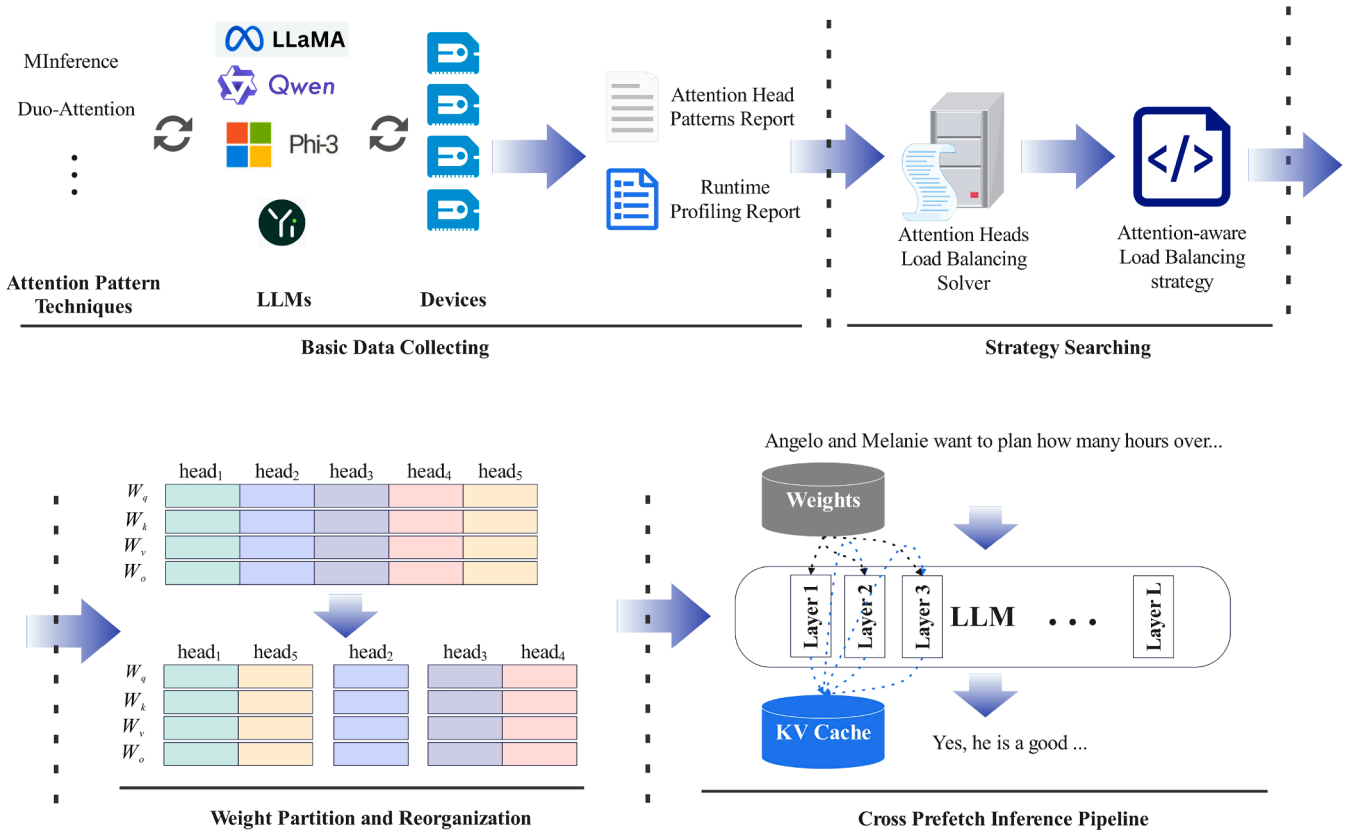


Fig. 3. The architecture of *FlexAttn*, comprises four core phases: basic data collection, strategy searching, weight partition & reorganization, and cross-prefetch inference pipeline.

Next, we explain the abovementioned approaches in detail in the following subsections.

4.2. Basic data collection

To search effective load-balancing strategies for long-context inference scenarios, relevant data support is essential to inform the search process. Therefore, we established a foundational data collection mechanism to support the derivation of inference strategies across varying text lengths. Specifically, we first analyzed the target LLM using the required long-context sparse technique to obtain the attention patterns for each head in every transformer layer, saved in the AHPR. Subsequently, as outlined in Algorithm 1, we profiled runtime performance across varying sequence lengths on target hardware devices to generate the runtime profiling report (RPR).

Following Algorithm 1, the basic data collection process consists of two main stages. First, we analyze each attention head in every layer of the LLM to identify its attention pattern based on the specified attention pattern technique. This information is recorded in the AHPR. The algorithm then identifies the unique set of attention patterns present in the model. Second, for each attention pattern type identified, we conduct runtime profiling across different sequence lengths from the sequence length set provided. The time costs measured are stored in the RPR, creating a comprehensive performance profile that maps each attention pattern and sequence length along with its corresponding computational cost. These collected data are an essential foundation for subsequent load-balancing optimization strategies.

To ensure compatibility with existing long-context inference techniques, we analyze sparse attention patterns. Based on state-of-the-art methods, including MInference [25], DuoAttention [26], Razor Attention [27], and RHME [28], we categorize attention patterns into four types: FULL, A-Shape, Vertical-Slash, and Block-Sparse. Table 2 presents the pattern adoption across these methods. The FULL pattern reflects standard self-attention computation. The A-Shape pattern, derived from StreamingLM [49], emphasizes initial tokens and recent context. The Vertical-Slash pattern combines vertical lines indicating global attention tokens with diagonal lines representing stepped local attention for information propagation. The Block-Sparse pattern focuses on contextually relevant segments within the current processing window.

Algorithm 1 Basic data collection.

Input: LLM $\mathcal{M}$, Attention Pattern Technique $\mathcal{T}$, Sequence Length Set $\mathcal{L}$
Output: AHPR, RPR

$AHPR \leftarrow \emptyset$, $RPR \leftarrow \emptyset$
$\mathcal{P} \leftarrow \emptyset$ ▷ Set of unique attention patterns
for layer l in $\mathcal{M}$ **do**
  for head h in layer l **do**
    pattern = AnalyzeAttentionPattern($\mathcal{T}$, h)
    $AHPR[l, h] \leftarrow$ pattern
    $\mathcal{P} \leftarrow \mathcal{P} \cap \{\text{pattern}\}$
  end for
end for
for pattern p in $\mathcal{P}$ **do**
  for length len in $\mathcal{L}$ **do**
    time_cost = ProfileRuntime(p, len)
    $RPR[p, len] \leftarrow$ time_cost
  end for
end for
return $AHPR$, RPR

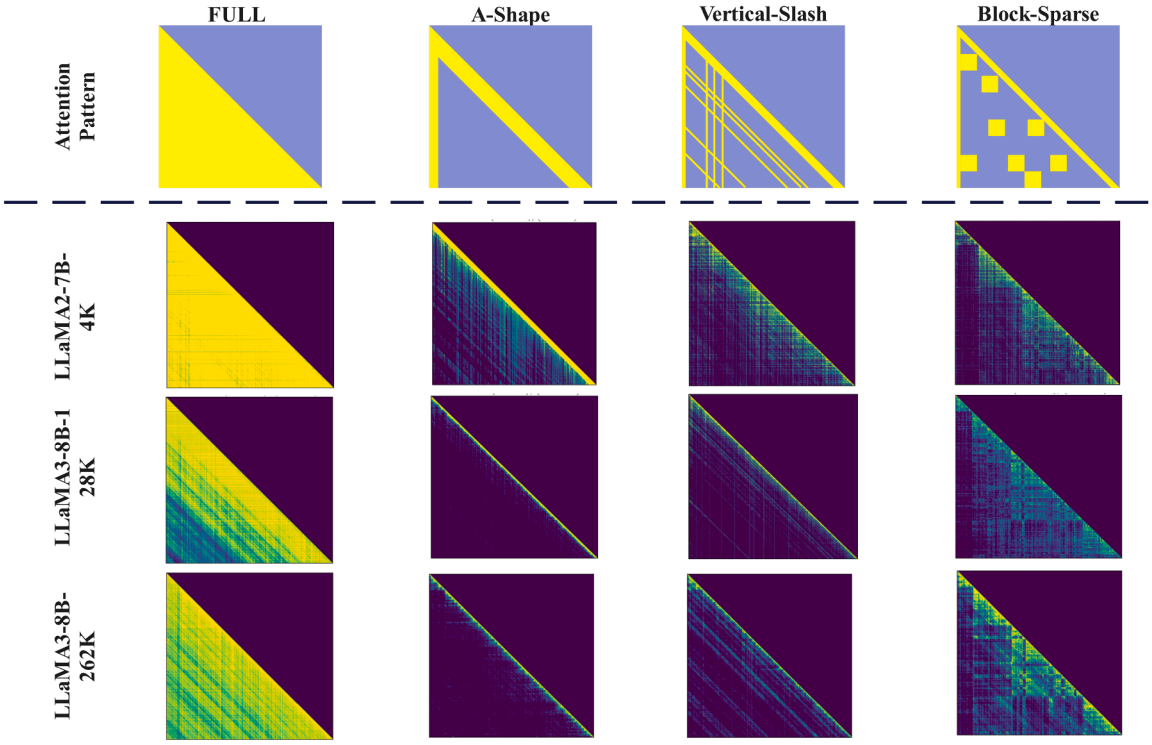


Fig. 4. Visualization of attention patterns across models.

Table 2
Attention pattern adoption in state-of-the-art (SOTA) methods.

Methods	FULL	A-Shape	Vertical-Slash	Block-Sparse
MInference		✓	✓	✓
DuoAttention	✓	✓		
RazorAttention	✓	✓		
RHME	✓	✓		

We visualize these four attention patterns using three representative models: LLaMA2-7B-4k,[2] LLaMA3-8B-128k,[3] and LLaMA3-8B-262k,[4] with relevant examples shown in Fig. 4. Heatmap-based visualizations elucidate the internal computational logic of different attention patterns, revealing their distinct computational characteristics. These empirically observed properties form the basis for the categorization implemented.

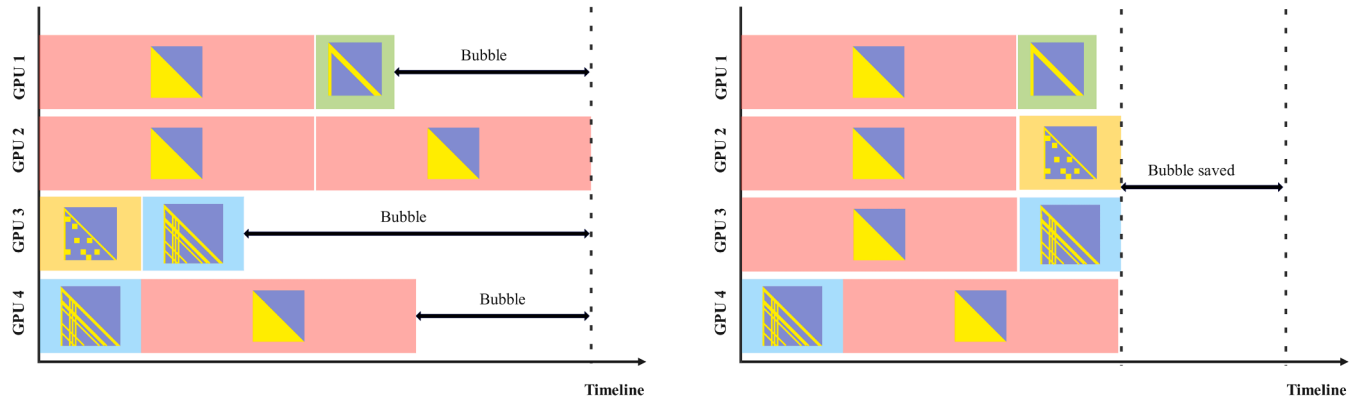
4.3. Strategy searching

In the backdrop of the continuous growth of sequence lengths, researchers have developed diverse attention patterns based on empirical observations from the performance of LLMs on practical data to address exponentially increasing computational demands. However, this approach alters the structural homogeneity inherent in the original multihead attention mechanism in the transformer and its operational paradigm, thus rendering uniform partitioning strategies such as Megatron [15] inapplicable in such scenarios.

We designed the attention-aware load-balancing strategy to achieve load balancing across devices for multiple attention heads within transformer layers. This strategy dynamically loads attention layer weights from the CPU to corresponding GPUs based on a preconfigured allocation and combination strategy. This strategy reduces device idle time (bubble) during execution (Fig. 5). Specifically, the uniform partitioning method of Megatron forces `GPU 1`, `GPU 3`, and `GPU 4` to wait for the completion of `GPU 2` before performing all-reduce operations, introducing bubble periods that waste computational resources and increase end-to-end latency. Through optimized attention head allocation enabled by our strategy (Fig. 5b), we achieve more balanced computational workloads across devices while reducing overall latency. In Fig. 6, we provide a further comparison between the tensor parallelism of Megatron-LM and our *FlexAttn*, demonstrating that the primary motivation of *FlexAttn* is to achieve more balanced GPU loads through overhead-aware allocation of different heads, thereby reducing GPU idle resource waste and ultimately improving inference efficiency. Furthermore, although the allocation of attention heads has been modified, the total amount of communication between multiple devices (which is only related to the sequence length and vector length) remains unaltered. As a result, our method does not impact the original communication pattern of the subsequent all-reduce operations.

We require a dedicated strategy-solving algorithm to search for a suitable attention-aware load-balancing strategy. This problem constitutes a typical NP-hard problem that could theoretically be addressed through combinatorial optimization methods. However, given that the number of attention heads typically reaches several dozen in practical scenarios, a precise solution becomes computationally prohibitive. Given practical constraints, we prioritize finding efficient solutions over achieving theoretical optimality. To this end, we propose an iterative optimization algorithm that can quickly converge to balanced GPU workload distributions across multiple GPUs while improving performance. We further develop a cost model as an evaluation proxy to enable strategy evaluation during the search process without dependence on physical hardware configurations, thereby avoiding computational resource constraints and prolonged optimization time.

[2] https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
[3] https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
[4] https://huggingface.co/gradientai/Llama-3-8B-Instruct-262k



(a) Naive partitioning causes GPU bubbles. (b) Balanced allocation minimizes idle time.

Fig. 5. Parallel strategy comparison for attention heads.

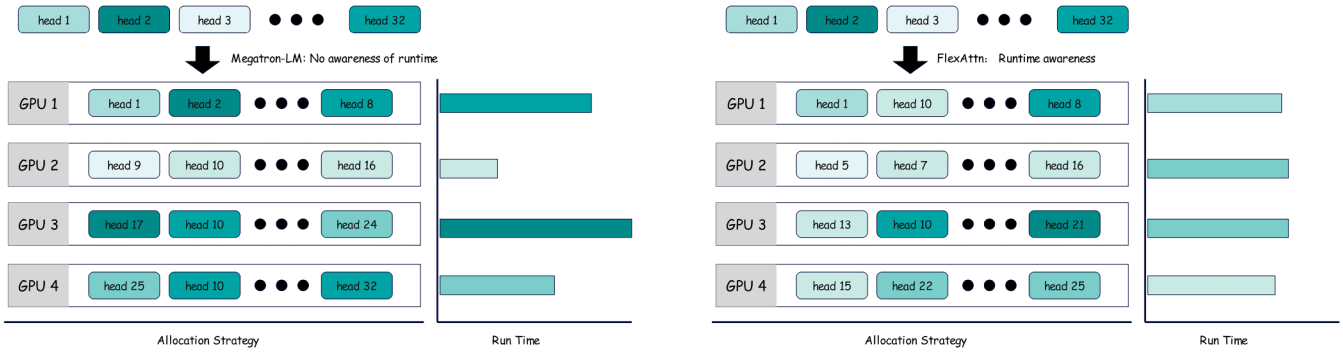


Fig. 6. Comparison between Megatron-LM tensor parallelism and *FlexAttn*. Megatron-LM adopts a static uniform allocation strategy without considering the actual computational overhead of each attention head; *FlexAttn* performs load-balanced allocation based on the time overhead of each attention head. Darker colors indicate higher computational overhead (illustrated with 32 attention heads).

4.3.1. Cost model

To construct a cost model, we first conduct an end-to-end comprehensive analysis of the LLM inference process. LLM architectures typically comprise three main components: an Embedding layer, Transformer layers, and a LM Head layer. The Transformer layers adopt a stacked structure, where multiple homogeneous Transformer layers are stacked to construct LLMs of different parameter scales, achieving varying cognitive capabilities. Each Transformer layer contains the attention layer and FFN/MLP layer (hereafter referred to as MLP layer) mentioned in Section 3 Preliminaries, along with two Layer Norm operations positioned before the attention and MLP operations, respectively.

Without considering single GPU memory constraints, the time cost model can be expressed as:

$$T^* = \arg\min_{S^*} \sum_{l=1}^{L} \left(T_{\text{norm}} + T_l^{\text{mha}}(S) + T_{\text{add}} + T_{\text{norm}} + T_{\text{mlp}} + T_{\text{add}}\right) + T_{\text{emb}} + T_{\text{norm}} + T_{\text{lm\_head}} \tag{7}$$

where T_{norm} denotes the Layer Norm computation cost, T_{add} is the addition of residuals. $T_l^{\text{mha}}(S)$ represents the multihead attention cost of layer l, T_{mlp} denotes the MLP operation cost within each layer, and T_{emb} and $T_{\text{lm\_head}}$ represent the costs of the Embedding and LM Head layer, respectively.

When GPU memory is limited and most data must be stored on the host side, and considering multi-GPU collaboration scenarios, we need to additionally consider: (1) communication overhead, such as loading weights from host memory to GPU and caching KV cache from GPU to CPU. (2) all-reduce overhead, such as tensor parallelism in Attention and MLP. The cost model can be further refined as follows:

$$T^* = \arg\min_{S^*} \sum_{l=1}^{L} \left(T_{\text{c2g}}^{\text{attn}} + T_{\text{norm}} + T_l^{\text{mha}}(S) + T_{\text{g2c}}^{\text{kv}} + T_{\text{all\_reduce}} + T_{\text{add}} + T_{\text{norm}} + T_{\text{c2g}}^{\text{mlp}} + T_{\text{mlp}} + T_{\text{all\_reduce}} + T_{\text{add}}\right) + T_{\text{c2g}}^{\text{emb}} + T_{\text{emb}} + T_{\text{norm}} + T_{\text{c2g}}^{\text{lm\_head}} + T_{\text{lm\_head}} \tag{8}$$

where $T_{\text{g2c}}^{\text{kv}}$, $T_{\text{c2g}}^{\text{attn}}$, $T_{\text{c2g}}^{\text{mlp}}$, $T_{\text{c2g}}^{\text{emb}}$, and $T_{\text{c2g}}^{\text{lm\_head}}$ represent the data transfer overhead among CPU with GPU for corresponding operations, and $T_{\text{all\_reduce}}$ denotes the all-reduce communication overhead of tensor parallelism. Eq. (8) can be used to estimate the overhead of LLM processing input text under strategy S^*.

After applying our proposed CPIP, we can leverage multi-process asynchronous parallel computing and communication to overlap the communication overhead with the computation overhead. In long-sequence scenarios where the computation overhead often significantly exceeds the communication overhead, $T_{\text{c2g}}^{\text{attn}}$, $T_{\text{c2g}}^{\text{mlp}}$, $T_{\text{c2g}}^{\text{emb}}$, and $T_{\text{c2g}}^{\text{lm\_head}}$ can be overlapped using parallel processing, effectively hiding their overhead. Consequently, our overall time overhead can be approximated by Eq. (9) instead of Eq. (8).

$$T^* = \arg\min_{S^*} \sum_{l=1}^{L} \left(T_{\text{norm}} + T_l^{\text{mha}}(S) + T_{\text{all\_reduce}} + T_{\text{add}} + T_{\text{norm}} + T_{\text{mlp}} + T_{\text{all\_reduce}} + T_{\text{add}}\right) + T_{\text{emb}} + T_{\text{norm}} + T_{\text{lm\_head}} \tag{9}$$

where we include $T_{\text{g2c}}^{\text{kv}}$ in $T_l^{\text{mha}}(S)$, because the data transfer is head independent and does not wait for all computations to finish in our design. This part will be elaborated in the subsequent Eq. (14).

Optimization Objective. Although the Eq. (9) can serve as an optimization objective, some terms are strategy-agnostic, and certain weights need not be entirely stored on the host. Specifically, the Embedding and LM Head layers can direct storage on the GPU to avoid redundant loading. Moreover, in long-context inference scenarios, the costs of MLP, Layer Norm, Addition, Embedding, and LM Head operations depend only on text length and parallelism strategy. The two all-reduce operations occur primarily after Attention and MLP operations, with costs related to text length and parallelism strategy, but not directly to our optimization objective. Therefore, the optimization objective can be simplified to:

$$T^* = \arg\min_{S^*} \sum_{l=1}^{L} \left(T_l^{\text{mha}}(S)\right) \tag{10}$$

For the above optimization objective, different Transformer layers in LLMs can be optimized independently. For a specific layer, the optimization objective can be expressed as follows:

$$T_l^{mha}(S) = \max_{d\in[1,D]} \left(\sum_{i\in\mathbb{D}_d} t_{l,i} \right), \tag{11}$$

where $t_{l,i}$ denotes the execution overhead of head i in layer l, and $\mathbb{D}_d$ represents heads allocated to device d. However, in practice, the computational overhead of each head is not independent but correlated with other heads in $\mathbb{D}_d$. This arises because modern LLM architectures increasingly adopt Grouped-Query Attention (GQA) [60] to reduce memory and computational costs associated with Key-Value (KV) caches. Specifically, only heads designated as query group leaders require full projection through W_Q, W_K, W_V, and W_O, while others share KV representations within their group and solely compute W_Q and W_O. Consequently, we decompose $t_{l,i}$ as follows:

$$t_{l,i} = \begin{cases} t_{l,i}^{attn} + t_{l,i}^{qkvo}, & \text{if } i \in \mathbb{V}_d \\ t_{l,i}^{attn} + t_{l,i}^{qo}, & \text{otherwise} \end{cases}, \tag{12}$$

where $t_{l,i}^{attn}$ denotes the intrinsic attention computation overhead, while $t_{l,i}^{qkvo}$ and $t_{l,i}^{qo}$ represent projection overhead with and without independent KV mappings, respectively. Here, $\mathbb{V}_d \subset \mathbb{D}_d$ comprises the minimal-indexed head from each query group within $\mathbb{D}_d$, which are responsible for full projections. To effectively reduce communication overhead in practical deployments, we adopt a priority-based processing strategy for $\mathbb{V}_d$ attention heads, enabling earlier transmission of KV caches to CPU memory. Therefore, the optimization objective can be transformed as follows:

$$T^* = \arg\min_{S^*} \sum_{l=1}^{L} \left(\max_d \Big(\sum_{i\in\mathbb{D}_d} \begin{cases} t_{l,i}^{attn} + t_{l,i}^{qkvo}, & \text{if } i \in \mathbb{V}_d \\ t_{l,i}^{attn} + t_{l,i}^{qo}, & \text{otherwise} \end{cases} + T_d^{g2c,kv} \Big) \right) \tag{13}$$

In Eq. (13), $T_d^{g2c,kv}$ represents the overhead of transferring KV cache from GPU to CPU. According to our design in *Section 4.5 Cross Prefetch Inference Pipeline*, the KV cache generated on each GPU only needs to be transferred before the next Transformer layer begins. Therefore, this overhead can be overlapped with Attention and MLP operations, and $T_d^{g2c,kv}$ is calculated as follows:

$$\begin{aligned} T_d^{g2c,kv} = \max\{ -(& \sum_{i\in\mathbb{D}_d} \begin{cases} t_{l,i}^{attn} + t_{l,i}^{qkvo}, & \text{if } i \in \mathbb{V}_d \\ t_{l,i}^{attn} + t_{l,i}^{qo}, & \text{otherwise} \end{cases} \\ & - t_{l,j}^{attn} - t_{l,j}^{qkvo} + T_d^{\text{mlp}} - \sum_{i\in\mathbb{V}_d} t_{kv}), 0 \}, \end{aligned} \tag{14}$$

where $t_{l,j}^{attn}$ and $t_{l,j}^{qkvo}$ represent the cost of the first head requiring KV cache computation (this portion of overhead cannot be overlapped), T_d^{mlp} denotes the MLP execution time on device d, and t_{kv} represents the overhead required to transfer KV cache of one head from GPU to CPU. The Eq. (14) indicates that if computation overhead can fully cover communication overhead, then $T_d^{g2c,kv}$ equals 0.

4.3.2. Strategy searching algorithm

We require a specialized strategy searching algorithm to obtain an appropriate load balancing strategy for multihead attention under given sequence lengths. This problem fundamentally belongs to the category of NP-hard combinatorial optimization, wherein theoretical approaches such as exhaustive search or complex combinatorial algorithms could be employed to search for global optimal solutions. However, each layer typically contains dozens of attention heads, enumerating all possible head allocation patterns or applying dynamic programming solutions would lead to exponential growth in computational complexity, making it infeasible to search for the optimal strategy within acceptable timeframes. To address this challenge, we employ an iterative optimization-based genetic algorithm (GA) for solution discovery, integrated with the previously established cost model to perform an offline evaluation of candidate strategies during the search process. This approach eliminates dependency on real hardware environments while avoiding requirements for prohibitively large-scale hardware resources and excessive computation time.

The feasible strategy for each layer can be represented as a 1D vector $\mathbf{S}_l = [s_{l,1}, s_{l,2}, \ldots, s_{l,h}]$ of length h, where h denotes the number of attention heads in that layer. Each element $\mathbf{S}_l[i] \in \{0, 1, \ldots, D-1\}$ specifies the device index assigned to the ith head for parallel computation. Here, each $\mathbf{S}_l$ corresponds to an individual in search process, while a composite strategy $\{\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_L\}$ represents a complete allocation scheme for multihead attention computation across all layers. To enhance efficiency, we execute the algorithm separately for each layer and select the optimal individual $\mathbf{S}_l^*$ with the best performance at the lth layer.

First, we need to generate the initial population for each layer l. Each individual $\mathbf{S}_l$ encodes the allocation scheme for h attention heads. To ensure population diversity, we generate N_p candidate strategies through uniform random sampling over $\{0, \ldots, D-1\}$. Formally, we define the search space $\mathbb{S}_l = \{0, 1, \ldots, D-1\}^h$. The initial population $P_l = \{\mathbf{S}_{l,1}, \mathbf{S}_{l,2}, \ldots, \mathbf{S}_{l,N_p}\}$ is generated by independently sampling each individual from this search space following a uniform distribution, i.e., $\mathbf{S}_{l,i} \overset{iid}{\sim} \text{Uniform}(\mathbb{S}_l)$. This ensures that each possible allocation pattern has an equal probability of being selected, providing comprehensive coverage of the solution space. Next, we will use the fitness function to evaluate the strategy. We quantify individual quality using the fitness function as follows:

$$F_l(\mathbf{S}_l) = -\max_d \Big(\sum_{i\in\mathbb{D}_d} t_{l,i} \Big), \tag{15}$$

where $t_{l,i}$ denotes the execution latency of the ith head at the lth layer, obtained from the RPR. Subsequently, the population enters an iterative replacement process to prevent the termination condition from being reached (e.g., maximum generations or fitness stagnation). Specifically, there are two operations involved here: crossover and mutation. For parent individuals $\mathbf{S}_l^{(p1)}$ and $\mathbf{S}_l^{(p2)}$, we employ multipoint crossover by splitting and exchanging subvectors at m positions $\alpha_1, \alpha_2, \ldots, \alpha_m$. Each offspring undergoes mutation with probability p_m, where $\mathbf{S}_l[i]$ is randomly reassigned to another device in $\{0, 1, \ldots, D-1\} \setminus \{\mathbf{S}_l[i]\}$. Offspring O_l and parents compete in a pooled candidate set, retaining the top N_p individuals by fitness. A small elite subset (current best solutions) is preserved to prevent the loss of global best strategies. The iterative "crossover-mutation-selection" cycle continues until the termination criteria are met.

The entire process is presented in Algorithm 2 for each layer of LLM. The function ¬TerminationCondition() in the third line represents a judgment on whether the stopping condition has been reached. The Crossover, Mutate, and Survive in the loop represent the crossover, mutation, and selection of the part of the strategy that survives the above-mentioned rule, respectively. This optimization balances computational workloads across devices, effectively minimizing synchronization overhead in multi-GPU environments.

Algorithm 2 Attention load balancing solver.

Input: Population size N_p, Search space $\mathbb{S}_l$, Fitness function F_l
Output: Optimal strategy $\mathbf{S}_l^*$

$P_l = \{\mathbf{S}_{l,1}, \mathbf{S}_{l,2}, \dots, \mathbf{S}_{l,N_p}\}$, $\mathbf{S}_{l,i} \overset{iid}{\sim} \text{Uniform}(\mathbb{S}_l)$, $i = 1, 2, \dots, N_p$
$f_{l,i} = F_l(\mathbf{S}_{l,i})$, $i = 1, 2, \dots, N_p$
while ¬TerminationCondition() **do**
  $O_l = \text{Crossover}(P_l)$ ▷ Generate offspring through crossover
  $O_l = \text{Mutate}(O_l)$ ▷ Apply mutation to offspring
  $f'_{l,i} = F_l(\mathbf{O}_{l,i})$, $i = 1, 2, \dots, |O_l|$
  $P_l = \text{Survive}(P_l \cup O_l, \{f_{l,i}\}_{i=1}^{N_p} \cup \{f'_{l,i}\}_{i=1}^{|O_l|}, N_p)$
  $\{f_{l,i}\}_{i=1}^{N_p} = F_l(\mathbf{S}_{l,i}), \mathbf{S}_{l,i} \in P_l$
end while
$\mathbf{S}_l^* \leftarrow \arg\max_{\mathbf{S}_l \in P_l} F_l(\mathbf{S}_l)$ ▷ Select best solution
return $\mathbf{S}_l^*$

4.4. Weight partition & reorganization

The conventional dynamic weight partition approach, which performs frequent decomposition $W_Q = [W_Q^0, W_Q^1, \dots, W_Q^h]$ followed by loading submatrices to corresponding GPUs through the attention load balancing strategy, suffers from two critical limitations. First, it generates frequent allocations of small memory chunks across GPUs, causing severe memory fragmentation. Second, repeated CPU-side matrix partitioning and extraction operations incur memory access overhead, ultimately leading to degraded runtime performance.

Our framework implements a prepartitioned weight organization strategy to address these challenges. We reorganize the original W_Q matrix into three logically grouped weight components $[W_Q^1, W_Q^5]$, $[W_Q^2]$, and $[W_Q^3, W_Q^4]$ based on the combined requirements (Fig. 3). These prepartitioned weights maintain contiguous memory storage to maximize spatial locality and minimize access latency.

This optimization improves hardware utilization efficiency and preserves strict mathematical equivalence with standard multihead attention computation as follows:

$$\mathbf{Q} = \mathbf{X}W_Q = \mathbf{X}[W_Q^0, \dots, W_Q^h] = [\mathbf{X}W_Q^0, \dots, \mathbf{X}W_Q^h] \tag{16}$$

where $W_Q^i \in \mathbb{R}^{d \times d_k}$ denotes the projection matrix of the ith head. Analogous decomposition applies to both $\mathbf{K}$ and $\mathbf{V}$ matrices. This decomposition enables the definition of atomic computation units $op_i^d = (\mathbf{X}|W_Q^i, W_K^i, W_V^i, W_O^i)$ for device d, the operations correspond to head i and do not change the original matrix computation and addition process.

4.5. Cross prefetch inference pipeline

The high-throughput-oriented zig-zag prefilling approach based on pipeline design ([32]) provides limited benefits in long-context scenarios where computational overhead exceeds I/O overhead. Therefore, building upon the pipeline design principles and their advantages, we tailor our approach to the characteristics of long-context inference scenarios (where computational overhead significantly exceeds weight loading overhead), and propose an alternative workflow (Fig. 7) that executes operations sequentially while leveraging asynchronous weight loading and Key-Value (KV) cache management to substantially reduce GPU memory pressure.

Our methodology decomposes each transformer layer into two computational phases: MHA and MLP. During MHA computation at layer l, three critical operations occur concurrently: (1) execution of the attention mechanism, (2) prefetching of MLP weights for the current layer, and (3) offloading of generated KV cache to CPU memory. Although our illustration shows KV cache management spanning the MHA and MLP phases, these operations are typically completed within the MHA phase for GQA-based models (Section 6.3). However, for LLMs with a larger number of KV heads, these operations may extend into the MLP phase. Thanks to the full-duplex communication architecture in modern hardware, data storage and loading operations proceed without blocking each other, while MLP computation at layer l proceeds in parallel with weight prefetching for MHA components at layer $l + 1$.

Based on the CPIP mechanism, we achieve finer-grained memory resource management within Transformer layers, enabling independent scheduling of attention and MLP module weights. In long-context inference scenarios, the bottleneck shifts from communication overhead to computational overhead, providing opportunities to optimize resource utilization through asynchronous communication. Specifically, taking the LLaMA3-8B as an example, under `torch.bfloat16` precision, the MHA and MLP modules require approximately 13GB of memory space in total (attention weights ~2.5GB, MLP weights ~10.5GB). Due to the significantly increased computational overhead of attention and MLP calculations in long-context scenarios, CPIP can independently schedule these two sub-modules without affecting subsequent layer computations. Through this design, each Transformer layer requires only about 0.33GB of active memory space, achieving pipeline inference and substantially reducing memory usage for weight storage. By overlapping I/O operations with computation, our *FlexAttn* effectively alleviates memory constraints, enabling larger models to run on limited hardware resources.

5. Experiments

FlexAttn aims to reduce prefilling latency and enhance overall throughput when processing long-context inputs in LLMs. Notably, our framework preserves the internal computations of the adopted sparse attention mechanisms as unchanged, and instead improves computational efficiency through attention scheduling from a resource utilization perspective. Therefore, our experimental evaluations primarily focus on efficiency metrics, as performance remains theoretically identical by design.

5.1. Metrics

This study employs two metrics to evaluate the performance of *FlexAttn*: (1) Latency (measured in seconds or milliseconds) quantifies end-to-end (E2E) inference delays or execution times of individual operators. (2) Throughput (tokens/second), calculated as the ratio of processed tokens to total execution time, measures data processing efficiency, with higher values indicating greater computational effectiveness.

5.2. Language models

Our experiments employ eight state-of-the-art long-context LLMs: Mistral-7B,[5] LLaMA3-8B-128K,[6] LLaMA3-8B-262K,[7] LLaMA3-8B-1048K,[8] ChatGLM4-9B-1M,[9] Yi-9B-200K,[10] Phi-3-Mini-128K,[11] and Qwen2-7B-32K.[12] For models with limited native context window support, we apply context extrapolation techniques (LM-Infinit [50]) to enable the processing of extended contexts.

The selected models exhibit diversity in four key aspects: (1) neural architectures, (2) pretraining data sources and methodologies, (3) native context length capacities, and (4) internal attention mechanisms. This heterogeneous combination ensures thorough validation of our proposed *FlexAttn* across multiple architectural and operational dimensions.

[5] https://huggingface.co/mistralai/Mistral-7B-v0.3
[6] https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
[7] https://huggingface.co/gradientai/Llama-3-8B-Instruct-262k
[8] https://huggingface.co/gradientai/Llama-3-8B-Instruct-Gradient-1048k
[9] https://huggingface.co/THUDM/glm-4-9b-chat-1m-hf
[10] https://huggingface.co/01-ai/Yi-9B-200K
[11] https://huggingface.co/microsoft/Phi-3-mini-128k-instruct
[12] https://huggingface.co/Qwen/Qwen2-7B-Instruct

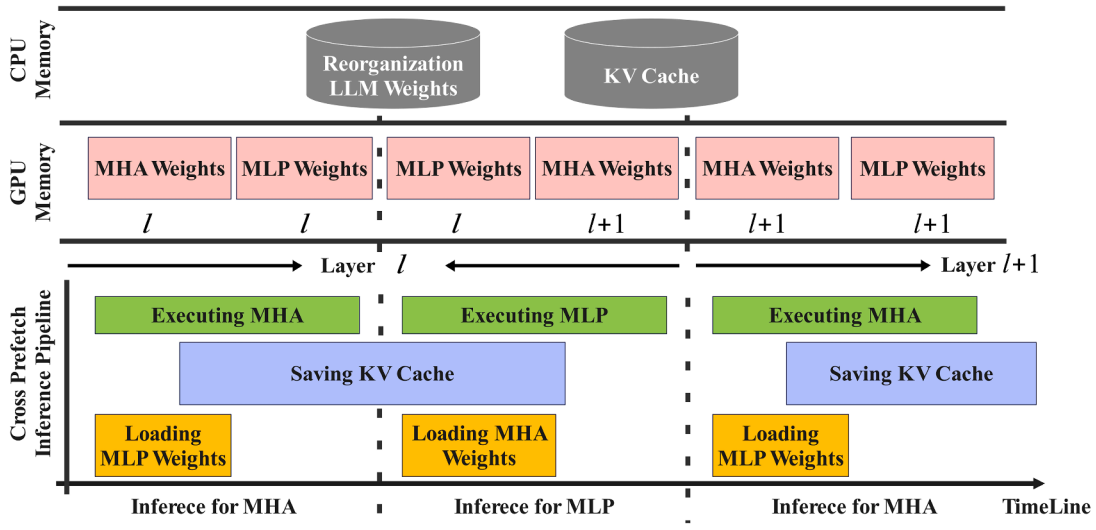


Fig. 7. Workflow of cross prefetch inference pipeline.

5.3. Baselines

We implement three baseline approaches for comparative analysis: (1) random allocation of attention heads across GPUs; (2) uniform partitioning (Megatron [15]), which sequentially distributes attention heads evenly across GPUs; and (3) random-uniform allocation, which maintains equal head counts per GPU through constrained randomization. To the best of our knowledge, this study represents the first systematic effort addressing computational imbalance in sparse attention mechanisms for long-text processing scenarios. Establishing these baselines ensures the validation of the effectiveness for *FlexAttn* through controlled comparisons.

5.4. Long-context techniques

This study implements two state-of-the-art long-context processing algorithms: MInference [25] and DuoAttention [26]. MInference incorporates three distinct attention patterns (A-Shape, Vertical-Slash, and Block-Sparse) to address significant latency during the prefill phase of LLMs in long-context scenarios. DuoAttention categorizes attention mechanisms into A-Shape and FULL patterns, employing a learnable gating mechanism to dynamically allocate computational resources between these operational modes based on contextual characteristics.

5.5. Implementation details

All LLMs and their corresponding weights were initialized using standard configurations from the HuggingFace library[13]. Our implementation leverages the PyTorch framework [61], incorporating Triton[14] and Flash Attention [62] optimized implementations to enable efficient inference across diverse attention patterns. The experiments were conducted on a server equipped with 4×NVIDIA RTX 3090 GPUs, featuring an Intel Xeon Gold 6130 CPU and 512GB RAM. The genetic algorithm employed in this study is configured with a population size of 30, a mutation rate of 0.02, single-point crossover (probability 0.8), a roulette wheel selection mechanism, and a maximum of 80 generations. The algorithm takes approximately 15–20 s in total on the LLaMA3-8B-262K model, with an average per-layer overhead of 0.46-0.625 s.

To evaluate performance across varying context lengths, we construct test data with controlled input dimensions. We adopt a literature-based question answering (QA) format for input data construction to assess method efficiency, eliminating the need for synthetic data generation techniques such as those employed in [63] for long-context synthesis. Our implementation utilizes Wikipedia as the external knowledge corpus [64][15], where each article is segmented into 100 semantically coherent passages. We establish a retrieval system using Elasticsearch[16] and implement a BM25 retriever to provide documents for each query.

In our experiments, we select 100 examples from the 2WikiMultihopQA dataset. For each query, the BM25 retriever identifies multiple relevant documents from Wikipedia, with the retrieval count dynamically adjusted according to predefined length requirements. To mitigate GPU warm-up effects on runtime measurements, we perform 20 executions per query under each target context length and calculate the mean value from the final 10 runs. Excluding the first 10 executions helps eliminate performance fluctuations that typically occur during the warm-up runs of the GPU. The prompt template is structured as follows:

Prompt Template

Please find relevant information from the literature based on the questions and answer the questions!
Question: When did John V, Prince Of Anhalt-Zerbst's father die?
Context:
[1672255] Christian August, Prince of Anhalt-Zerbst Christian August……
[12419160] Frederick Augustus, Prince of Anhalt-Zerbst Frederick Augustus, Prince……
…
[1672259] 1727 in Vechelde, Christian August married Johanna Elisabeth……
Question: When did John V, Prince Of Anhalt-Zerbst's father die?
So the answer is:

[13] https://huggingface.co/
[14] https://github.com/triton-inference-server/
[15] https://dl.fbaipublicfiles.com/dpr/wikipedia_split/psgs_w100.tsv.gz
[16] https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.17.9-linux-x86_64.tar.gz

Table 3
Based on different long-context techniques, the efficiency with different lengths is compared under different LLMs.

Length	LMs	Random		Uniform		Random-Uniform		*FlexAttn*	
		E2E (s) ↓	Thr.↑	E2E (s) ↓	Thr. ↑	E2E (s)↓	Thr. ↑	E2E (s) ↓	Thr. ↑
768K					MInference				
	LLaMA3-8B-262K	492.89	1595.52	412.21	1907.81	435.82	1804.48	**321.37**	**2447.07**
	Phi-3-Mini-128k	351.81	2235.37	312.27	2518.35	319.90	2458.34	**214.70**	**3662.90**
	Qwen2-7B-32K	464.54	1692.92	393.59	1998.05	392.45	2003.87	**327.62**	**2400.40**
	Yi-9B-200k	536.66	1465.40	484.43	1623.39	490.47	1603.39	**370.20**	**2124.32**
	ChatGLM4-9B-1M	783.83	1003.31	631.33	1245.67	660.20	1191.19	**572.43**	**1373.83**
					DuoAttention				
	LLaMA2-7B-32K	989.37	794.87	868.35	905.65	871.18	902.71	**646.59**	**1216.26**
	LLaMA3-8B-1048k	953.18	825.05	934.02	841.97	837.11	939.45	**608.65**	**1292.08**
	Mistral-7B	958.03	820.87	981.56	801.20	833.49	833.49	**610.31**	**1288.56**
160K					MInference				
	LLaMA3-8B-262K	48.57	3373.18	36.49	4489.43	44.85	3652.96	**28.29**	**5791.02**
	Phi-3-Mini-128k	27.06	6054.23	24.74	6620.96	24.69	6634.50	**19.73**	**8300.14**
	Qwen2-7B-32K	33.95	4825.24	28.59	5728.68	33.15	4942.17	**23.10**	**7090.17**
	Yi-9B-200k	59.05	2774.29	48.64	3367.98	56.41	2904.18	**39.67**	**4129.75**
	ChatGLM4-9B-1M	55.71	2940.46	45.36	3611.93	52.65	3111.72	**37.59**	**4358.08**
					DuoAttention				
	LLaMA2-7B-32K	72.70	2253.57	63.73	2570.47	63.74	2570.22	**39.72**	**4124.34**
	LLaMA3-8B-1048k	62.30	2629.75	54.34	3014.88	58.73	2789.69	**38.56**	**4248.30**
	Mistral-7B	63.57	2577.31	55.71	2940.92	57.83	2833.05	**38.59**	**4244.80**

5.6. Main results

To evaluate the performance of our proposed method in processing ultra-long sequences, we conducted comprehensive experiments comparing it with three established attention head allocation approaches: Random, Uniform, and Random-Uniform. We test these approaches across different language models under two sequence length configurations (768K and 160K tokens), with detailed results presented in Table 3. Our *FlexAttn* demonstrates strong performance across all configurations, showing significant advantages in accelerating ultra-long-context inference tasks.

At the 768K sequence length, while adopting either MInference or DuoAttention techniques, our *FlexAttn* achieves the shortest end-to-end latency and highest throughput. For instance, with LLaMA3-8B-262K under MInference, our approach achieves 321.37 s end-to-end latency, which is 34.8 %, 22.0 %, and 26.3 % faster than the Random, Uniform, and Random-Uniform methods, respectively. The throughput reaches 2447.07 tokens/s, exceeding other methods by 53.4 %, 28.3 %, and 35.6 %, respectively. Under DuoAttention with LLaMA3-8B-1048K, our method reduces end-to-end latency by 36.2 %, 34.8 %, and 27.3 % compared to the baselines while improving the throughput by 56.6 %, 53.5 %, and 37.5 %, respectively. These results demonstrate the efficiency of *FlexAttn* in handling ultra-long sequences. For 160K-length sequences, our *FlexAttn* maintains performance advantages. With ChatGLM4-9B-1M under MInference, we achieve 37.59 s end-to-end latency (32.5 % and 17.1 % faster than Random and Uniform, respectively) and 4358.08 tokens/s throughput (48.2 % and 20.7 % higher than Random and Uniform, respectively).

A comparative analysis reveals greater efficiency gains at 768K sequences, aligning with our theoretical analysis shown in Fig. 2. Longer sequences exacerbate load imbalance issues, and our *FlexAttn* adapts well to these varying overhead across different sequence lengths through optimized GPU workload distribution.

Notably, efficiency variations were observed across models. Phi-3-Mini-128K demonstrates better performance under MInference owing to its smaller parameter scale, while larger models like ChatGLM4-9B-1M and Yi-9B-200k exhibit higher computational overhead. Nevertheless, our *FlexAttn* consistently outperforms baselines across all models, demonstrating robust applicability. Extensive experimental evidence confirms the effectiveness of *FlexAttn* in accelerating long context inference through attention head load balancing, particularly excelling in extreme-length scenarios while maintaining best throughput and latency metrics.

Table 4
Performance comparison across different layers with context length 768k. Execution times (s) are reported for layer-wise computations and end-to-end (E2E) processing for the whole LLM.

LC. Algorithms	Methods	Layer 5	Layer 15	Layer 25	E2E
MInference	Uniform	13.48	14.84	7.36	412.21
	FlexAttn	**10.56**	**12.26**	**5.82**	**321.37**
DuoAttention	Uniform	19.94	35.43	34.47	868.35
	FlexAttn	**19.02**	**22.93**	**19.01**	**646.59**

5.7. Ablation studies

5.7.1. Performance analysis of FlexAttn into layers

To gain deeper insights into the advantages offered by *FlexAttn* in processing ultra-long contexts, we conducted a detailed layer-wise comparison between the MInference and DuoAttention algorithms before and after applying *FlexAttn*. We randomly selected three layers (5th, 15th, and 25th) in the LLaMA3-8B-1048k model with a fixed context length of 768K tokens, with results summarized in Table 4.

With MInference, *FlexAttn* achieves consistent acceleration across layers. At layer 5, the execution time reduces from 13.48s to 10.56s (21.7 % speedup). At layer 15, the runtime reaches 12.26s (17.4 % faster than Uniform); at layer 25, the execution time is 5.82s (20.9 % improvement). End-to-end analysis shows a total runtime reduction from 412.21s to 321.37s, achieving a 22.0 % speedup, demonstrating stable efficiency gains across all layers with the MInference technique. In DuoAttention, *FlexAttn* achieves more substantial improvements. While layer 5 shows a modest 4.6 % acceleration (19.02s vs. Uniform's 19.94s), layers 15 and 25 exhibit remarkable 35.3 % and 44.8 % reductions, respectively. The cumulative effect yields 25.5 % end-to-end acceleration (646.59s vs. 868.35s). Moreover, for Layer 5 under DuoAttention, we observed that the uniform strategy achieves similar performance to

Table 5
Effect of different components on *FlexAttn* performance (throughput, tokens/s).

Methods	LLaMA3	Phi-3	Qwen2	Yi	ChatGLM4
Uniform	1907.81	2518.35	1998.05	1623.39	1245.67
Ours w/o WPR	2434.77	3607.48	2383.12	2114.06	1370.09
Ours w/o CPIP	2114.27	2866.29	2204.65	1852.08	1279.81
Ours	**2447.07**	**3662.90**	**2400.40**	**2124.32**	**1373.83**

FlexAttn, which can be attributed to the relatively even distribution of FULL attention patterns in this layer, with the patterns of kv heads 1, 2, 4, and 7 are FULL, each of them allocated to one GPU. Additionally, extra overhead costs are associated with KV cache storage and weight loading, resulting in an execution time approximately 1 s longer than that of *FlexAttn*.

By comparing MInference and DuoAttention, we discovered that *FlexAttn* achieves a more significant speedup on DuoAttention, primarily because DuoAttention employs FULL and A-Shape attention patterns, which have substantial differences in computational costs. *FlexAttn* adapts well to these inherent computational differences between attention patterns of different heads by dynamically adjusting attention head allocation across layers, achieving more balanced execution times.

5.7.2. Effectiveness analysis of different components

Table 5 presents the effectiveness analysis of different optimization components in *FlexAttn*. This experiment adopts the MInference long-text algorithm with a context length of 768K, maintaining consistency with the model selection specified in Table 3 for MInference. We evaluate the performance variations when removing two key components: weight partition and reorganization (WPR) and cross prefetch inference pipeline (CPIP). We compare them with both the complete *FlexAttn* framework and the Uniform baseline.

The results demonstrate that *FlexAttn* achieves the highest throughput across all models, outperforming the Uniform method by 10.3 % to 45.4 %. This validates its effectiveness in enhancing ultra-long text inference efficiency. When removing WPR, the throughput shows minor degradation (0.3 %-1.5 % across models), indicating that WPR stably improves performance by avoiding CPU resource competition caused by dynamic weight partition and reorganization during inference. We hypothesize that WPR could provide more substantial benefits in edge scenarios with constrained CPU resources. In Section 6.8 Performance of WPR across Different CPUs, we conduct some extended analyses to further investigate this hypothesis, thereby warranting further exploration in the future.

The removal of CPIP leads to significant performance degradation, with throughput reduction ranging from 6.8 % to 21.7 % across models. For instance, LLaMA3 exhibits a 13.6 % throughput drop (from 2447.07 to 2114.27 tokens/s) without CPIP. This strongly demonstrates CPIP's critical role in *FlexAttn*, where its cross-layer prefetching and pipeline execution mechanisms effectively overlap the communication overhead with computation processes, thereby substantially boosting the inference performance.

We also present the performance of our method, which relies solely on the proposed attention load balancing solver (ALBS) and weight partition & reorganization (WPR), against other baselines. We removed the cross prefetch inference pipeline (CPIP) component from the *FlexAttn*. The specific results are shown in Table 6.

As shown in the experimental results in Table 6, even without the CPIP, *FlexAttn*† still demonstrates significant performance improvements. Specifically, on the LLaMA3-8B-262K model, compared to the uniform method, *FlexAttn*† reduces the end-to-end inference time from 412.21 s to 371.96 s (a 9.7 % reduction), and increases throughput from 1907.81 to 2114.27 (a 10.8 % improvement). On the Phi-3-Mini-128k model, compared to the uniform method, *FlexAttn*† reduces inference time from 312.27 s to 274.37 s (a 12.1 % reduction), and increases throughput from 2518.35 to 2866.29 (a 13.8 % improvement). On the Qwen2-7B-32K model, compared to the random-uniform method, *FlexAttn*† reduces inference time from 392.45 s to 356.72 s (a 9.1 % reduction), and increases throughput from 2003.87 to 2204.65 (a 10.0 % improvement). On the Yi-9B-200k model, compared to the uniform method, *FlexAttn*† reduces inference time from 484.43 s to 424.62 s (a 12.3 % reduction), and increases throughput from 1623.39 to 1852.08 (a 14.1 % improvement). On the ChatGLM4-9B-1M model, compared to the uniform method, *FlexAttn*† reduces inference time from 631.33 s to 614.49 s (a 2.7 % reduction), and increases throughput from 1245.67 to 1279.81 (a 2.7 % improvement). These results demonstrate that the ALBS and WPR components we proposed can bring significant performance improvements without CPIP.

To validate the effectiveness of the ALBS(+ WPR) components from another perspective, we further construct a stronger baseline, Uniform + CPIP. As shown in Table 7, *FlexAttn* demonstrates superior performance even when compared against this enhanced baseline. Specifically, *FlexAttn* reduces the time overhead on the five tested models by 17.3 %, 13.0 %, 14.2 %, 20.5 %, and 6.9 %, respectively. These results conclusively demonstrate that our proposed ALBS offers significant advantages, effectively balancing the load across different GPUs for LLMs utilizing the sparse inference technique in long-context inference scenarios, thereby improving resource utilization and ultimately reducing end-to-end computational overhead.

5.7.3. Scalability analysis across sequence lengths

Fig. 8 compares the performances of different attention head allocation methods across varying sequence lengths on the LLaMA3-8B-1048K model. We evaluate five sequence lengths (256K, 384K, 512K, 640K, and 768k) while recording end-to-end inference time and throughput. The results demonstrate that all methods exhibit increased inference time and reduced throughput in the case of longer sequences owing to amplified computational and communication costs. Notably, *FlexAttn* consistently achieves the shortest inference time and highest throughput across all lengths. At 768K sequence length, *FlexAttn* attains 321.37 s inference time, outperforming Random, Uniform, and Random-Uniform by 34.8 %, 22.0 %, and 26.3 %, respectively, with corresponding throughput improvements of 53.4 %, 28.3 %, and 35.6 %, respectively. For 384K sequences, *FlexAttn* reduces inference time by 34.5 %, 21.5 %, and 26.1 % compared to baseline methods, accompanied by throughput gains of 52.6 %, 27.4 %, and 35.3 %, respectively. Even at shorter sequences (256k), *FlexAttn* delivers 20.8 % - 35.8 % inference acceleration and 26.7 % - 55.7 % throughput enhancement. These results validate the superior efficiency of *FlexAttn* across diverse lengths.

Table 8 compares *FlexAttn* with Uniform across multiple language models at shorter sequence lengths (140K - 200K). Evaluations on five models reveal the consistent throughput superiority of *FlexAttn*. For ChatGLM4-9B-1M, *FlexAttn* achieves 4409.19 tokens/s throughput at 140K length, exceeding Uniform by 21.8 %, with a 19.6 % advantage at 200K. The performance gap becomes more pronounced in Phi-3-Mini-128k, showing 39.4 %-70.8 % throughput improvements. *FlexAttn* achieved throughput gains of 26.6 % - 30.3 %, 22.3 % - 25.5 %, and 21.2 % - 22.6 % for LLaMA3-8B-262K, Qwen2-7B-32K, and Yi-9B-200k respectively across various lengths. These findings confirm the stable superiority of *FlexAttn* over the conventional uniform strategy across various models and moderate sequence lengths.

Fig. 8 and Table 8 reveal that *FlexAttn* demonstrates remarkable efficiency advantages for long sequences (>512K) while maintaining consistent superiority across different language models.

6. Analysis

6.1. Does it affect the output results?

The proposed method does not alter the inherent computational logic of existing sparse attention techniques, and theoretical analysis suggests

Table 6
Efficiency comparison analysis on different models. *FlexAttn*† denotes the version without the CPIP. The experiments use a text length of 768k and employ the MInference sparse attention technique.

LMs	Random		Uniform		Random-Uniform		*FlexAttn*†	
	E2E (s) ↓	Thr.↑	E2E (s) ↓	Thr. ↑	E2E (s)↓	Thr. ↑	E2E (s) ↓	Thr. ↑
LLaMA3-8B-262K	492.89	1595.52	412.21	1907.81	435.82	1804.48	**371.96**	**2114.27**
Phi-3-Mini-128k	351.81	2235.37	312.27	2518.35	319.90	2458.34	**274.37**	**2866.29**
Qwen2-7B-32K	464.54	1692.92	393.59	1998.05	392.45	2003.87	**356.72**	**2204.65**
Yi-9B-200k	536.66	1465.40	484.43	1623.39	490.47	1603.39	**424.62**	**1852.08**
ChatGLM4-9B-1M	783.83	1003.31	631.33	1245.67	660.20	1191.19	**614.49**	**1279.81**

Table 7
Performance comparison between *FlexAttn* and Uniform + CPIP on different models (text length: 768k, sparse attention technique: MInference). Values outside parentheses indicate latency, while values in parentheses represent throughput (tokens/s).

Methods	LLaMA3-8B-262K	Phi-3-Mini-128k	Qwen2-7B-32K	Yi-9B-200k	ChatGLM4-9B-1M
Uniform + CPIP	388.69(2023.25)	248.66(3162.63)	382.05(2058.46)	465.84(1688.21)	614.83(1279.10)
FlexAttn	321.31(2447.51)	216.30(3635.74)	327.62(2400.41)	370.20(2124.32)	572.43(1373.83)

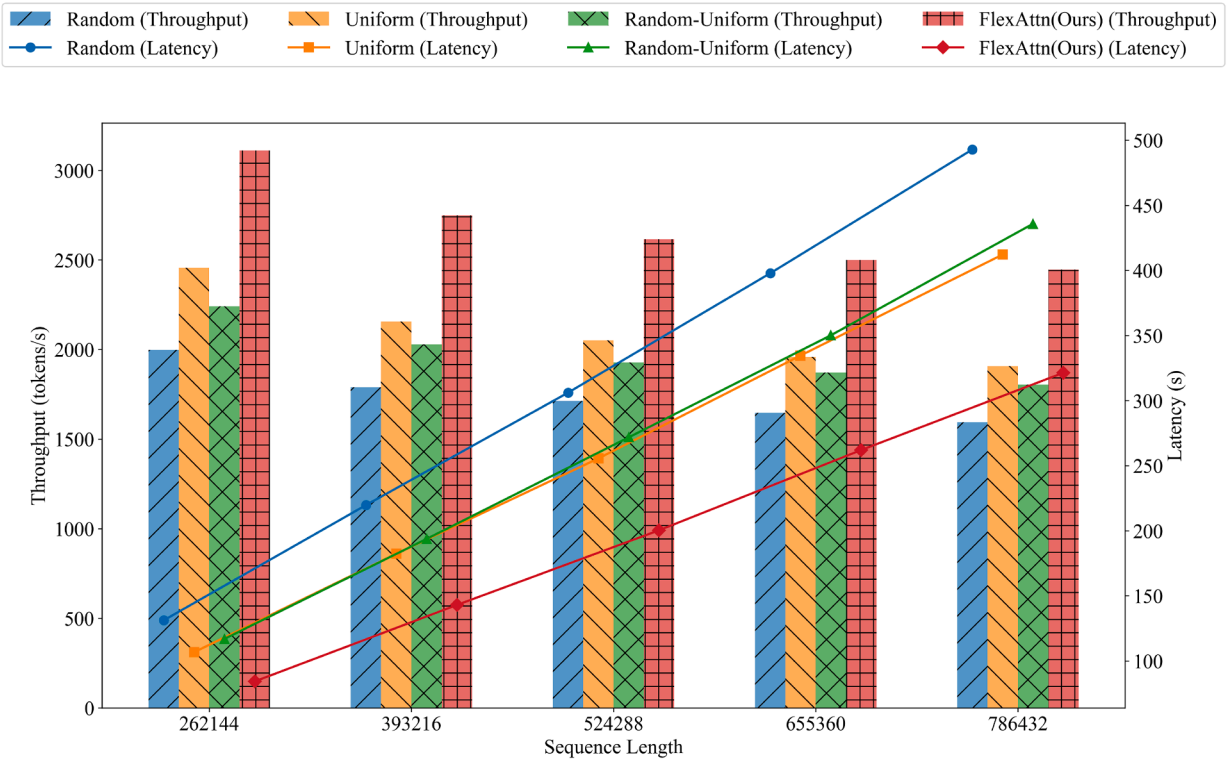


Fig. 8. Comparison of performance at different lengths, based on the LLaMA3-8B-1048K model.

Table 8
Comparison of inference efficiency (throughput tokens/s) with different lengths (≤ 200k) based on MInference.

LMs	Methods	140k	160k	180k	200k
LLaMA3-8B-262K	Uniform	4458.83	4489.43	4653.95	4717.49
	FlexAttn	**5811.38**	**5791.02**	**5928.73**	**5969.75**
Phi-3-Mini-128k	Uniform	5659.93	6620.96	4842.88	4881.91
	FlexAttn	**7891.09**	**8300.14**	**8273.05**	**8319.24**
Qwen2-7B-32K	Uniform	5381.16	5728.68	5745.34	5826.11
	FlexAttn	**6754.10**	**7090.17**	**7059.12**	**7127.82**
Yi-9B-200k	Uniform	3356.86	3367.98	3452.71	3494.11
	FlexAttn	**4115.01**	**4129.75**	**4207.73**	**4234.70**
ChatGLM4-9B-1M	Uniform	3620.87	3611.93	3708.45	3778.66
	FlexAttn	**4409.19**	**4358.08**	**4459.18**	**4517.97**

that results obtained using our *FlexAttn* should be equivalent to those from the original method. To validate this claim, Table 9 presents experimental results obtained on the Infinite Bench dataset [65], which contains diverse tasks designed to evaluate the capabilities of LLMs in processing long-context data. Using the LLaMA3-8B-262K model with greedy decoding to ensure reproducibility, our experiments set the maximum context length to 160k tokens and randomly sampled 50 instances per subtask.

In Table 9, the evaluation metrics for different tasks are specified as follows: Math.Find, Code.Debug, En.Dia, En.MC, Retr.PassKey, and Retr.Number use accuracy as the metric. Zh.QA and En.QA employ ROUGE F1 scores. En.Sum utilizes ROUGE-L Sum scores.

The experimental results demonstrate that our *FlexAttn* did not affect model accuracy, confirming that our approach preserves the orig-

Table 9
Results across different tasks under MInference.

Methods	En.Sum	En.QA	EN.MC	EN.Dia	Zh.QA	Code.Debug	Math.Find	Retr.PassKey	Retr.Number	Avg.
without *FlexAttn*	21.27	9.75	70.00	8.00	12.99	16.00	92.00	100.00	100.00	47.78
with *FlexAttn*	21.27	9.75	70.00	8.00	12.99	16.00	92.00	100.00	100.00	47.78

Table 10
Experimental configurations (Notations: F = FULL, A = A-Shape, V = Vertical-Slash, B = Block-Sparse; → = sequential ordering; seq. = sequential groups; ×N = repetition factor; + = combination).

Scenario	GPUs	Heads	Attention Arrangement
S1	2	16	8F → 8A.
S2	2	16	4 × (F, A, V, B) seq.
S3	2	16	4-type random.
S4	4	32	16F → 16A.
S5	4	32	8 × (F, A, V, B) seq.
S6	4	36	12 × (A, V, B) seq.
S7	4	32	4-type random.
S8	4	32	F+A random.
S9	4	72	A+V+B random.
S10	8	72	4-type random.

inal logic of sparse attention computation and achieving performance equivalent to those of conventional implementations.

6.2. Performance under different GPU configurations

To comprehensively validate the effectiveness of our proposed method and accommodate potential future variations in attention head configurations within LLMs, we design extended experimental scenarios with diverse attention pattern allocations. Through these set scenarios, we simulate the distribution of various attention patterns that may appear in the future so as to further explore the applicability of the method proposed in this paper. For these experiments, we examine a single transformer layer since the optimization of each layer is independent and multiple layers simply represent a stack of similar optimizations. We construct the experimental layer based on LLaMA3-8B architecture. The GPU configurations include an 8-GPU setup (deployed on a high-cost system with 768GB RAM and 8 NVIDIA 3090 GPUs). We establish 10 distinct experimental scenarios covering variations in GPU quantities, attention head type permutations, and total head counts. Comparative analysis across these scenarios enables evaluation of our optimization allocation strategy. Detailed configurations are presented in Table 10.

To evaluate the performance of *FlexAttn* across various GPU counts, we conducted experiments on scenarios S3, S7, and S10 from Table 10, with results shown in Fig. 9. Fig. 9 presents a comparative analysis of runtime distribution across GPUs for our *FlexAttn* under different parallel configurations (S3, S7, and S10), benchmarked against Uniform and Random baseline methods.

In S3 (Fig. 9a), Uniform exhibits significant runtime variance among various GPUs (2.32s vs 4.00s, 42.08 % difference), while *FlexAttn* maintains near-identical execution times across devices (0.22 % variation). The results in S7 (Fig. 9b) reveal an exacerbated imbalance in baseline methods with 4 GPUs. Uniform shows an extreme disparity between GPU1 (0.70s) and GPU3 (5.45s, 87.14 % difference), and the Random method maintains 58.90 % variation. In contrast, *FlexAttn* achieves tightly clustered runtimes (3.62s-3.68s, 1.57 % difference). The configuration S10 (Fig. 9c) with 8 GPUs highlights the superiority of *FlexAttn* over baselines demonstrating substantial imbalances (84.70 % and 86.77 % differences for Uniform and Random, respectively). Notably, Uniform and Random present 6-fold runtime disparities between the fastest/slowest GPUs (e.g., 0.82s vs 5.34s), whereas *FlexAttn* maintains a maximum variation of 28.56 % between GPU6 and GPU7.

Our method consistently reduces inter-GPU runtime disparities across configurations, enabling balanced workload distribution. This contrasts with baseline approaches that fail to mitigate computational heterogeneity, resulting in underutilized devices and degraded system efficiency. The empirical results validate the robustness of *FlexAttn* in multi-GPU parallel computing, which optimally allocates computational loads to maximize hardware utilization, thereby accelerating inference and improving throughput.

Fig. 10 compares experimental results across multiple scenarios defined in Table 10, where we validate method robustness through diverse parameter combinations and stochastic configurations. The empirical results demonstrate that our approach consistently achieves superior end-to-end latency reduction across all scenarios evaluated.

6.3. Visualization of inference

To demonstrate the manner in which our framework operates in practice, we conduct inference on a 768k-context input and visualize its detailed execution process, including computational overhead and parallel scheduling in specific layers of the LLaMA3-8B model in Fig. 11.

The execution trace shows that CPIP enables the overlapping of Key-Value cache storage with attention computation. Our approach prioritizes computing heads that require shared Key-Value pairs on each GPU, allowing immediate data transfer to the CPU. After completing attention computation, an all-reduce operation is performed, followed by RMSNorm (with weights stored in GPU memory). During MLP computation, we prefetch attention weights for the 7-*th* layer based on the load balancing strategy. Due to varying numbers of heads assigned to each GPU, the weight loading requirements differ, resulting in varying prefetch durations across GPUs (with GPU1 exhibiting the longest latency). After MLP computation, another round of All-Reduce operations is required to ensure all GPU synchronization.

As previously mentioned, current models like LLaMA3-8B employ GQA technology [60], resulting in some heads sharing KV pairs. In the figure, white blocks represent these shared KV heads, while red blocks indicate heads requiring storage. For instance, GPU1 handles 10 attention heads but only performs one GPU-CPU transfer for head 25. The visualization demonstrates that our method achieves relatively balanced attention computation overhead across four GPUs.

6.4. Impact on all-reduce and computation-communication competition

We present a detailed analysis of the relationship between our *FlexAttn* and all-reduce operations, as well as whether computation-communication resource competition exists.

As shown in Fig. 12, all-reduce operations exhibit strictly serial characteristics in the computational flow. **All-reduce after Attention**: Occurs after the `o_proj` operation, which maps the 128-dimensional features of each attention head to the hidden state dimension (4096 in this example). **All-reduce after MLP**: Taking the LLaMA model as an example, the MLP layer contains three projection operations: `up_proj`, `gate_proj`, and `down_proj`. The `up_proj` and `gate_proj` partition the weight matrix column-wise, while `down_proj` partitions the weight matrix row-wise. The all-reduce operation takes place after the `down_proj` operation.

These two all-reduce operations are executed serially after the `o_proj` and `down_proj` projection operations, respectively, thus no resource competition exists between computation and communication. Therefore, the primary factors affecting all-reduce operations are the

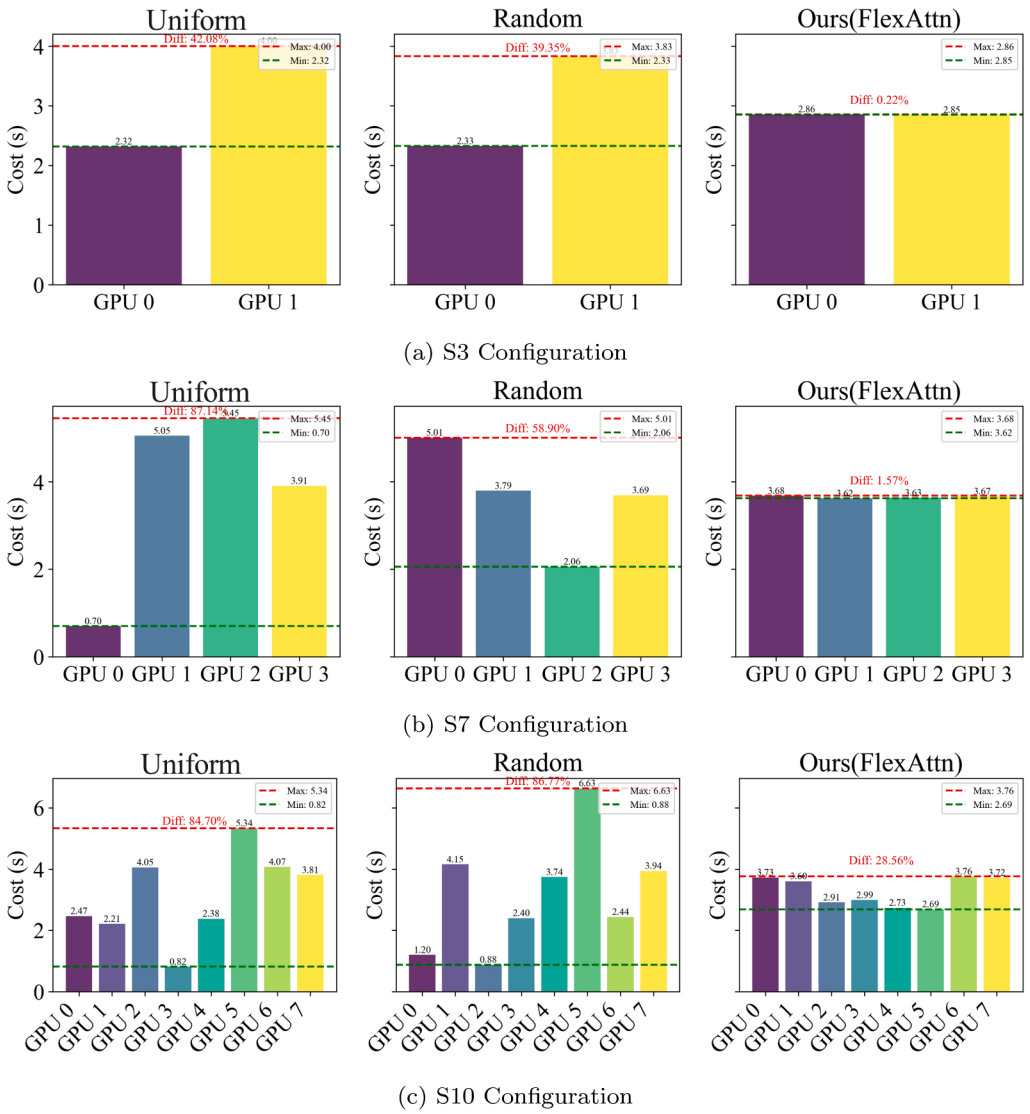


Fig. 9. Comparative analysis of computational costs using LLaMA3-8B under different GPU configurations.

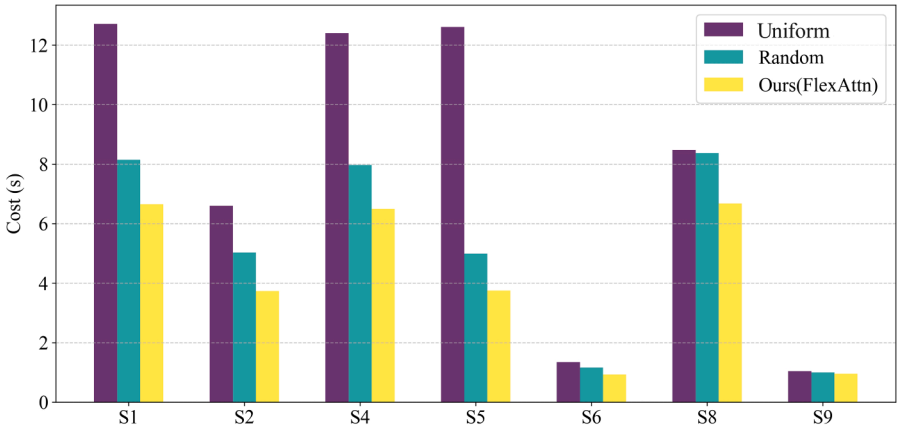


Fig. 10. Performance comparison under different scenarios.

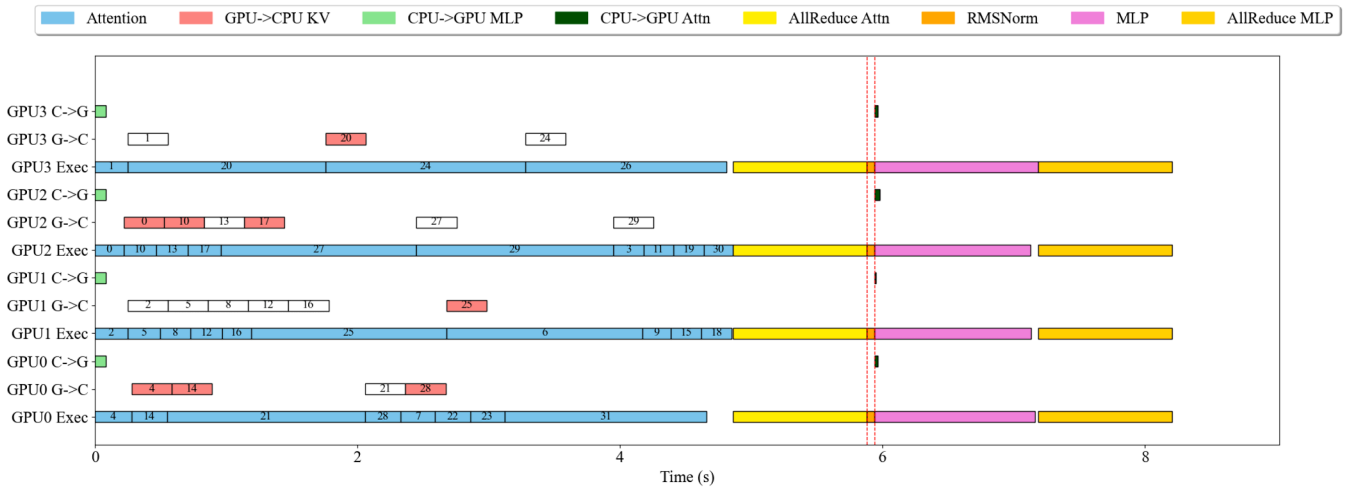


Fig. 11. Execution details of 6-*th* layer in LLaMA3-8B with 768k context length.

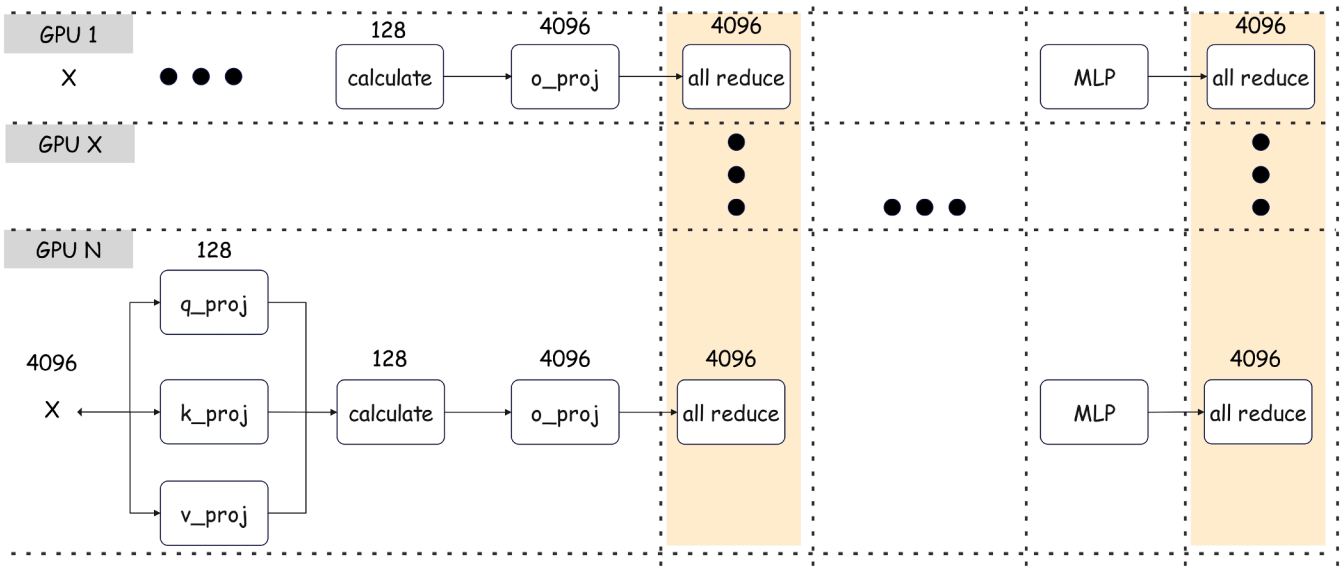


Fig. 12. Visualization of the all-reduce operations inside the transformer layer, involving multiple GPUs. The hidden state (x) has a feature length of 4096 and consists of 32 heads, each head has a feature length of 128.

number of tokens (i.e., input text length), GPU configuration, and bandwidth. The above analysis demonstrates that our method does not affect the All-Reduce overhead required by the LLM framework itself, and our scheduling does not cause all-reduce to compete with Attention or MLP or communications of other modules.

6.5. Does the runtime of the attention mechanism affect the initiation of subsequent operations?

Fig. 13 presents an in-depth view of how varying sequence lengths, which directly affect the runtime of the attention mechanism, can influence the initialization overhead for subsequent operations. As the context length increases, the attention mechanism requires more computation, leading to a potentially longer waiting time before the next stage, such as all-reduce or other layer computations, can begin. Despite this slight delay, the results show that the influence of the initialization overhead (the run time cost interval between the completion of one operation and the initiation of the next) remains on the order of 10^{-6} to 10^{-7} s, which is markedly small relative to the overall inference overhead, and can be neglected in practical scenarios.

As the sequence length increases, although the overhead of the attention mechanism increases, it has almost no impact on the initialization of subsequent operations.

6.6. Cost model effectiveness analysis

To verify the accuracy of the proposed cost model in reflecting the execution characteristics of actual systems, we conducted comparative experiments using the LLaMA3-8B-262K model. These experiments aimed to validate the generalization ability of the cost model under different scenarios.

Experimental Setup: We conduct tests with three different context lengths (128k, 256k, and 384k). For each context length, the experimental process is as follows: (1) Randomly generate 10 sets of different attention head allocation strategies, with each strategy independently allocating attention heads to different GPU devices for each layer of the model; (2) For each strategy, use the cost model to calculate theoretical execution time, considering the costs of all key components including attention computation, QKV projection, output projection, and inter-GPU communication; (3) Execute LLM inference tasks under the correspond-

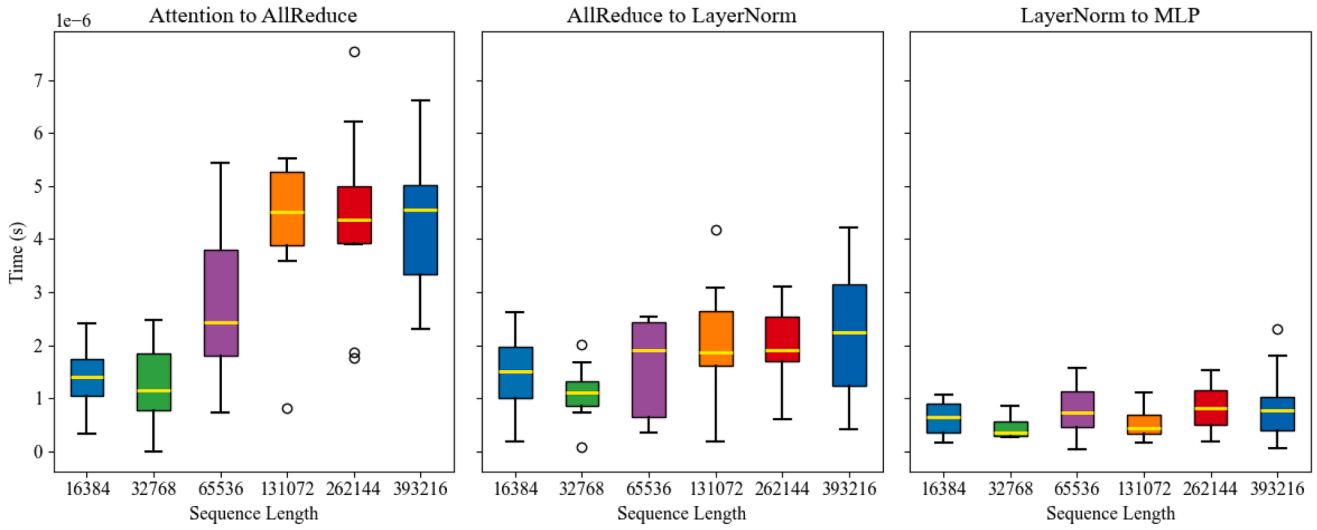
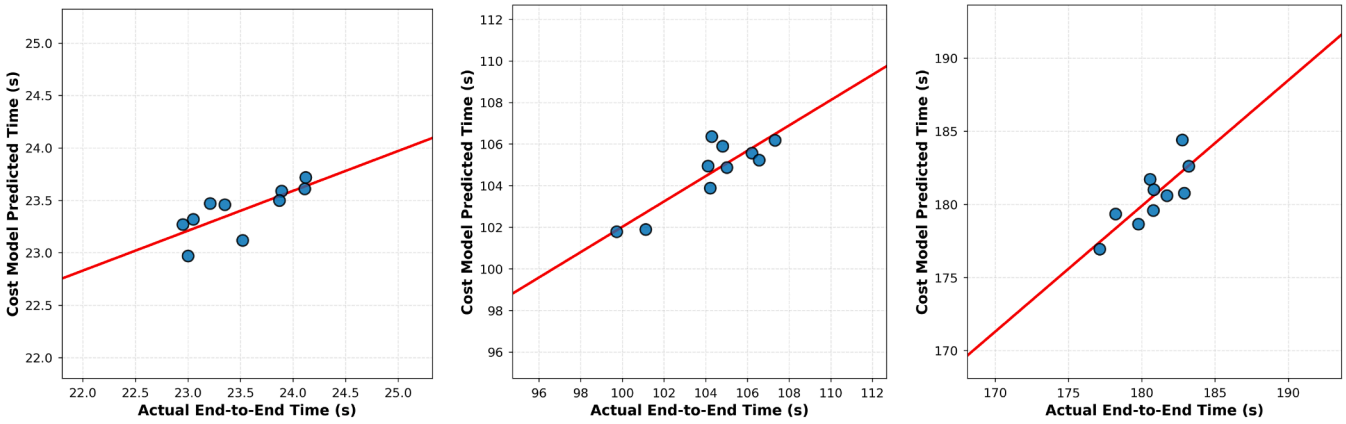


Fig. 13. Impact of sequence length on the initialization overhead of subsequent operations.



(a) Context length of 128k (b) Context length of 256k (c) Context length of 384k

Fig. 14. Cost model validation results for the LLaMA3-8B model at three different context lengths. The horizontal axis represents actual end-to-end execution time, the vertical axis represents cost model estimated time, and the straight line represents the linear fitting of data points.

ing strategy on the actual system, record end-to-end execution time, and repeat each strategy 10 times, taking the average cost to eliminate.

Result Analysis: Fig. 14 shows the comparison results between the cost model estimated overhead and the actual execution overhead. With different context lengths, the cost model estimates and actual execution overhead show strong correlation, indicating that the model can accurately capture execution characteristics. As context length increases, the distribution of data points becomes more concentrated, because longer contexts make computation time dominate overall execution time, reducing the relative impact of system noise. The strong correlation demonstrates that this cost model can effectively guide the algorithm to search for reasonable allocation strategies.

The minor discrepancies between the cost model and actual execution time mainly stem from: operating system scheduling, memory management, and actual bandwidth system requisition for communication. Despite these uncontrollable factors, the experimental results demonstrate that the proposed cost model has sufficient correlation to effectively guide the genetic algorithm in searching efficient attention head allocation strategies.

6.7. Comparative analysis of optimization algorithms

The selection of an appropriate optimization algorithm for the attention head assignment problem requires careful consideration. While alternative approaches such as sophisticated greedy algorithms or simulated annealing are viable candidates, we selected the genetic algorithm based on empirical evaluation and theoretical considerations. Our choice is primarily motivated by the adaptability to varying problem scales and its robust performance of GA across different LLM architectures. As LLMs continue to evolve with potentially larger numbers of attention heads, we think the scalability of GA is relatively superior. Nevertheless, the final choice should depend on specific application scenarios. In our initial design phase, we also considered dynamic programming approaches, but due to severe computational efficiency issues, we ultimately abandoned this method in favor of heuristic approaches such as genetic algorithms and simulated annealing.

Specifically, the greedy algorithm employs a KV-aware greedy strategy that sorts attention heads by computation time in descending order and uses a minimum-increment allocation principle, for each head, it

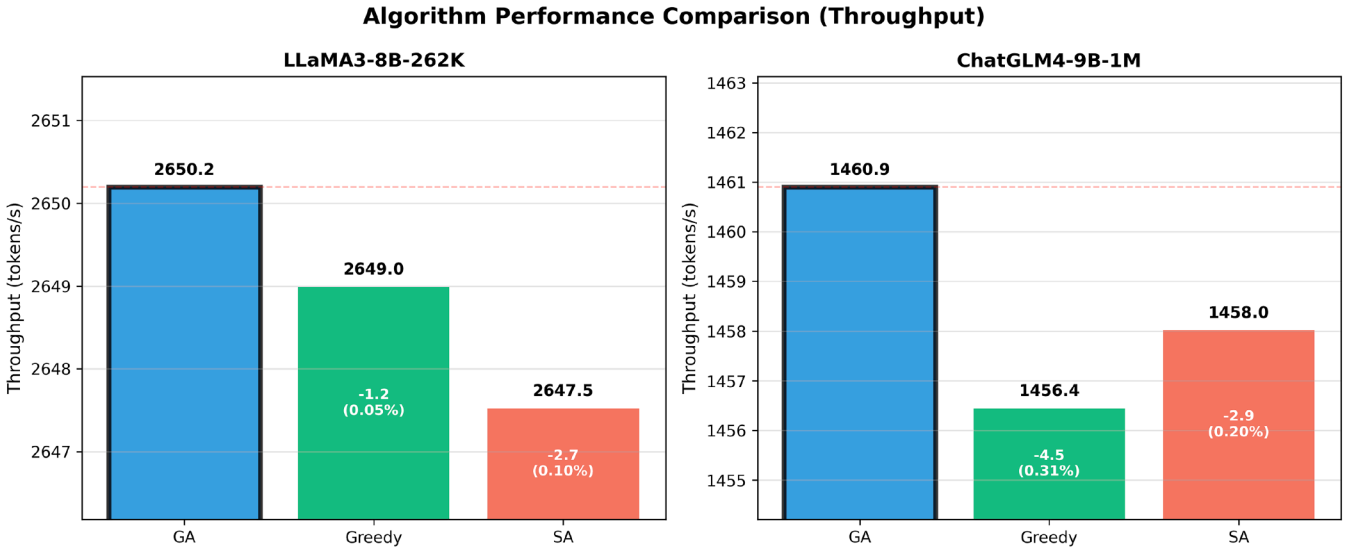


Fig. 15. Throughput comparison of GA (Genetic Algorithm), Greedy (Greedy Algorithm), and SA (Simulated Annealing) using LLaMA3-8B-262K and ChatGLM4-9B-1M models with context length is 512K.

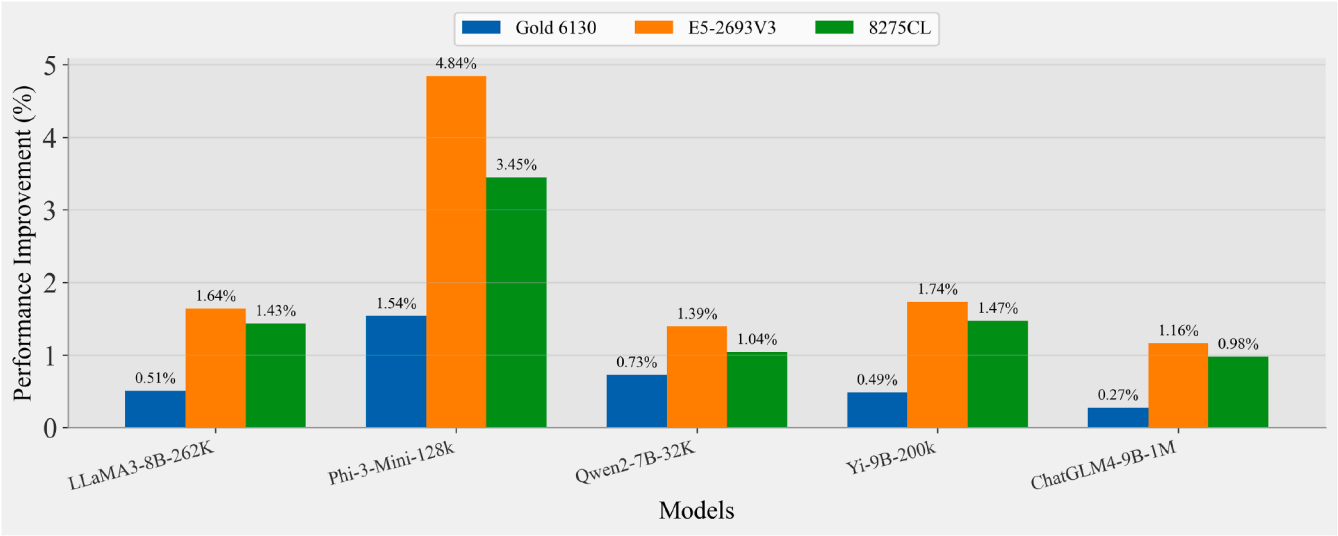


Fig. 16. Performance improvement by WPR across different CPUs, with a 768K context length.

evaluates the incremental cost of assignment to each GPU (considering KV-sharing: if a GPU already contains heads from the same group, only `qo_time` is required. Otherwise, the full `qkvo_time` is needed), and selects the assignment that minimizes the increase in system-wide maximum load. The Simulated Annealing algorithm defines three neighborhood operations (swap: exchanging the assignments of two heads, move: relocating a single head, group_swap: exchanging entire KV groups) and uses the Metropolis criterion for state transitions. The specific parameter settings for the Genetic Algorithm (GA) remain consistent with those in the experimental section.

Fig. 15 presents a comparative analysis of the three algorithms' performance. The genetic algorithm consistently achieves the highest throughput across both LLM models. Notably, while the simulated annealing algorithm underperforms the greedy approach on the LLaMA model, GA maintains stable and superior performance across different architectures. Considering the overall performance and stability of GA, we choose it as our preferred algorithm for attention head allocation optimization.

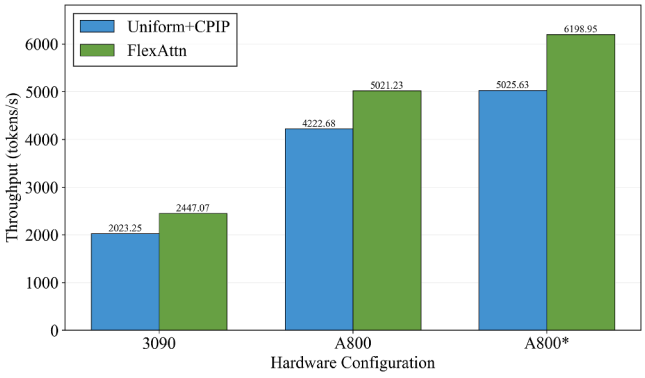


Fig. 17. Performance comparison under different computing devices and communication hardware scenarios. 3090 represents 4 NVIDIA RTX 3090 GPUs, A800 represents 4 NVIDIA RTX A800 GPUs, and A800* represents A800 GPUs with NVLink.

6.8. Performance of WPR across different CPUs

We conducted supplementary experiments using three different CPUs, including `Intel Xeon Gold 6130` (our baseline configuration), `Intel Xeon E5-2693V3`, and `Intel Xeon 8275CL`.

As shown in Fig. 16, when deploying LLMs to process long-context content on relatively older devices, the partitioning and reorganization of weights requires more time, thereby impacting overall performance. It can be observed that the Phi-3-Mini model is more significantly affected, and consequently, after applying weight partition and reorganization (WPR) on the E5-2693V3 CPU, the throughput improvement reaches 4.84 %. Overall, on the machine with the E5-2693V3 CPU, WPR can deliver an average throughput improvement of 2.15 %. On the 8275CL machine, it can provide an average enhancement of 1.67 %. This further confirms our initial expectations for the WPR approach during the design phase, demonstrating that leveraging WPR can yield corresponding benefits in scenarios where CPU resources are more limited. Since the all-reduce operation is also considerably affected on the E5-2693V3 and 8275CL machines, if we further correct the impact of all-reduce based on the data from the 6130 CPU, the average improvement can reach 3.10 % and 2.73 %, respectively.

These results confirm the practical significance of WPR in real-world deployments, especially in cost-sensitive scenarios where organizations need to leverage existing infrastructure or deploy models on edge devices. This holds important practical value for application scenarios where frequent hardware upgrades are not feasible or where decommissioned legacy equipment needs to be repurposed while still requiring deployment of advanced language models.

6.9. Performance analysis on different hardware platforms

To comprehensively evaluate the applicability and robustness of *FlexAttn*, this section explores the performance under different hardware configurations. We particularly focus on the impact of GPU computing capability and interconnect technology (PCIe vs. NVLink). We used the LLaMA3-8B-262k model in a 768k text length inference task to compare the performance of *FlexAttn* (ALBS + WPR + CPIP) with a baseline method (Uniform + CPIP), based on the MInference sparse attention technique.

The results are shown in Fig. 17. The experimental results demonstrate that *FlexAttn* exhibits stable performance advantages across different environments, with throughput improvements ranging from 18.91 % to 23.35 %. On the RTX 3090 platform, *FlexAttn* achieves a throughput of 2447.07 tokens/s, representing a 20.95 % improvement over the baseline method's 2023.25 tokens/s. On the A800 platform, *FlexAttn* achieves an 18.91 % performance improvement. With NVLink equipped, *FlexAttn* still maintains a 23.35 % throughput improvement. These results demonstrate the stability and effectiveness of *FlexAttn* across different hardware environments. Meanwhile, we observe that with improved communication hardware performance (such as using NVLink), the acceleration of all-reduce operations indeed brings significant overall throughput improvement, which is consistent with our theoretical analysis.

These findings provide preliminary support for the applicability of *FlexAttn* in practical deployments. As hardware and LLM technologies evolve, we recommend that practitioners adapt the configurations of our approach based on their specific hardware capabilities, LLM architectures, and deployment requirements to achieve optimal performance.

7. Conclusion

This study presents *FlexAttn*, an automated attention load balancing framework for long-context inference, designed to address the computational load imbalance in large language models (LLMs) when combined with sparse attention techniques. This framework comprises four coordinated phases: basic data collection, strategy searching, weight partition & reorganization, and cross prefetch inference pipeline. Our *FlexAttn* dynamically optimizes attention workload distribution across transformer layers during long-context inference, significantly enhancing computational efficiency. We integrate *FlexAttn* with state-of-the-art long-context inference methods, including MInference and DuoAttention, conducting extensive evaluations across various LLM architectures. Experimental results demonstrate significant performance improvements, achieving a throughput improvement of 34.95 % to 40.9 % on LLaMA3-8B across context lengths ranging from 160k to 768k tokens. Notably, our framework remains orthogonal to traditional model parallelism techniques and existing sparse attention implementations, enabling complementary performance gains when combined with them. These advancements position *FlexAttn* as a novel and effective solution for overcoming computational challenges in the long-context inference of LLMs, with promising implications for practical applications across diverse domains.

8. Limitations and future work

Although our method enhances computational efficiency through attention load balancing, we acknowledge certain practical limitations. Current implementation constraints restrict context length extensions to 768k tokens on four 3090 GPUs under limited hardware configurations. We anticipate growing research interest in long-context processing as large language models (LLMs) gain prominence, particularly regarding 1 million tokens context that future applications may increasingly demand. Our subsequent work will prioritize developing resource-efficient solutions for 1 million tokens inference, addressing both algorithmic challenges and computational optimization in constrained environments. Additionally, we will optimize all-reduce communication for contexts exceeding 1 million tokens. The core approach involves deeply fusing all-reduce operators with `o_proj` and `down_proj` operators, achieving overlapping execution of communication and computation through blocked computation, thereby effectively reducing the overall latency of LLM inference.

CRediT authorship contribution statement

Jie Ou: Writing – review & editing, Writing – original draft, Software, Methodology, Data curation, Conceptualization; **Jinyu Guo:** Writing – review & editing, Writing - original draft, Methodology, Conceptualization; **Shuaihong Jiang:** Writing – review & editing, Software, Data curation; **Xu Li:** Writing – review & editing, Software, Data curation; **Ruini Xue:** Writing – review & editing, Supervision, Conceptualization; **Wenhong Tian:** Writing – review & editing, Methodology, Conceptualization; **Rajkumar Buyya:** Writing - review & editing, Supervision, Conceptualization.

Data availability

Data will be made available on request.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This research is supported by the Sichuan International Science and Technology Innovation Cooperation Project with ID 2024YFHZ0317, the Chengdu Science and Technology Bureau Project with ID 2024-YF09-00041-SN, the National Natural Science Foundation of China Project with ID W2433163, and the Postdoctoral Fellowship Program (Grade C) of China Postdoctoral Science Foundation with ID GZC20251053.

References

[1] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H.W. Chung, C. Sutton, S. Gehrmann, et al., Palm: scaling language modeling with pathways, J. Mach. Learn. Res. 24 (240) (2023) 1–113.

[2] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al., Llama: Open and efficient foundation language models, (2023). arXiv preprint arXiv:2302.13971

[3] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F.L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al., Gpt-4 technical report, (2023). arXiv preprint arXiv:2303.08774

[4] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, et al., The LLAMA 3 herd of models, (2024). arXiv preprint arXiv:2407.21783

[5] K. Shuang, Z. Zhouji, W. Qiwei, J. Guo, Thinking about how to extract: energizing LLMs' emergence capabilities for document-level event argument extraction, in: Findings of the Association for Computational Linguistics ACL 2024, 2024, pp. 5520–5532.

[6] J. Guo, X. Chen, Q. Xia, Z. Wang, J. Ou, L. Qin, S. Yao, W. Tian, HASH-RAG: bridging deep hashing with retriever for efficient, fine retrieval and augmented generation, in: W. Che, J. Nabende, E. Shutova, M.T. Pilehvar (Eds.), Findings of the Association for Computational Linguistics: ACL 2025, Association for Computational Linguistics, Vienna, Austria, 2025, pp. 26847–26858. https://doi.org/10.18653/v1/2025.findings-acl.1376

[7] Z. Zeng, Q. Cheng, X. Hu, Y. Zhuang, X. Liu, K. He, Z. Liu, KoSEL: knowledge subgraph enhanced large language model for medical question answering, Knowl. Based Syst. 309 (2025) 112837.

[8] B. Li, S. Fan, S. Zhu, L. Wen, CoLE: a collaborative legal expert prompting framework for large language models in law, Knowl. Based Syst. 311 (2025) 113052.

[9] R. Bairi, A. Sonwane, A. Kanade, A. Iyer, S. Parthasarathy, S. Rajamani, B. Ashok, S. Shet, Codeplan: repository-level coding using llms and planning, Proc. ACM Software Eng. 1 (FSE) (2024) 675–698.

[10] C. Li, Z. Zheng, X. Du, X. Ma, Z. Wang, X. Li, KnowBug: enhancing large language models with bug report knowledge for deep learning framework bug prediction, Knowl. Based Syst. 305 (2024) 112588.

[11] A. Caciularu, M.E. Peters, J. Goldberger, I. Dagan, A. Cohan, Peek across: improving multi-document modeling via cross-document question-answering, (2023). arXiv preprint arXiv:2305.15387

[12] T. Li, G. Zhang, Q.D. Do, X. Yue, W. Chen, Long-context LLMS struggle with long in-context learning, (2024). arXiv preprint arXiv:2404.02060

[13] J. Schulman, B. Zoph, C. Kim, J. Hilton, J. Menick, J. Weng, J.F.C. Uribe, L. Fedus, L. Metz, M. Pokorny, et al., ChatGPT: optimizing language models for dialogue, OpenAI blog (2022).

[14] J. Ou, J. Guo, S. Jiang, Z. Wang, L. Qin, S. Yao, W. Tian, Accelerating adaptive retrieval augmented generation via instruction-driven representation reduction of retrieval overlaps, in: W. Che, J. Nabende, E. Shutova, M.T. Pilehvar (Eds.), Findings of the Association for Computational Linguistics: ACL 2025, Association for Computational Linguistics, Vienna, Austria, 2025, pp. 26983–27000. https://doi.org/10.18653/v1/2025.findings-acl.1384

[15] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, B. Catanzaro, Megatron-LM: Training multi-billion parameter language models using model parallelism, (2019). arXiv preprint arXiv:1909.08053

[16] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q.V. Le, Y. Wu, et al., Gpipe: efficient training of giant neural networks using pipeline parallelism, , Adv. Neural Inf. Process. Syst. 32, 2019, pp. 103–112.

[17] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, E.P. Xing, Poseidon: an efficient communication architecture for distributed deep learning on GPU clusters, in: 2017 USENIX Annual Technical Conference (USENIX ATC 17), 2017, pp. 181–193.

[18] H. Cui, H. Zhang, G.R. Ganger, P.B. Gibbons, E.P. Xing, Geeps: scalable deep learning on distributed gpus with a GPU-specialized parameter server, in: Proceedings of the Eleventh European Conference on Computer Systems, 2016, pp. 1–16.

[19] H. Liu, M. Zaharia, P. Abbeel, RingAttention with blockwise transformers for near-infinite context, in: The Twelfth International Conference on Learning Representations, 2024.

[20] Z. Li, S. Zhuang, S. Guo, D. Zhuo, H. Zhang, D. Song, I. Stoica, Terapipe: token-level pipeline parallelism for training large-scale language models, in: International Conference on Machine Learning, PMLR, 2021, pp. 6543–6552.

[21] V.A. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, B. Catanzaro, Reducing activation recomputation in large transformer models, Proc. Mach. Learn Syst. 5 (2023) 341–353.

[22] C. Holmes, M. Tanaka, M. Wyatt, A.A. Awan, J. Rasley, S. Rajbhandari, R.Y. Aminabadi, H. Qin, A. Bakhtiari, L. Kurilenko, et al., Deepspeed-fastGEN: High-throughput text generation for LLMS via mii and deepspeed-inference, (2024). arXiv preprint arXiv:2401.08671

[23] D. Li, H. Wang, E. Xing, H. Zhang, AMP: automatically finding model parallel strategies with heterogeneity awareness, in: Advances in Neural Information Processing Systems, 35, 2022, 6630–6639.

[24] L. Zheng, Z. Li, H. Zhang, Y. Zhuang, Z. Chen, Y. Huang, Y. Wang, Y. Xu, D. Zhuo, E.P. Xing, et al., Alpa: automating inter-and intra-Operator parallelism for distributed deep learning, in: 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22), 2022, pp. 559–578.

[25] H. Jiang, Y. Li, C. Zhang, Q. Wu, X. Luo, S. Ahn, Z. Han, A. Abdi, D. Li, C.-Y. Lin, et al., Minference 1.0: accelerating pre-filling for long-context LLMS via dynamic sparse attention, in: Advances in Neural Information Processing Systems,37, 2025, 52481–52515.

[26] G. Xiao, J. Tang, J. Zuo, J. Guo, S. Yang, H. Tang, Y. Fu, S. Han, Duoattention: efficient long-context LLM inference with retrieval and streaming heads, (2024). arXiv preprint arXiv:2410.10819

[27] H. Tang, Y. Lin, J. Lin, Q. Han, S. Hong, Y. Yao, G. Wang, Razorattention: efficient KV cache compression through retrieval heads, (2024). arXiv preprint arXiv:2407.15891

[28] W. Wu, Y. Wang, G. Xiao, H. Peng, Y. Fu, Retrieval head mechanistically explains long-context factuality, (2024). arXiv preprint arXiv:2404.15574

[29] L. Ren, Y. Liu, S. Wang, Y. Xu, C. Zhu, C.X. Zhai, Sparse modular activation for efficient sequence modeling, in: Advances in Neural Information Processing Systems,36, 2023, 19799–19822.

[30] A. Han, J. Li, W. Huang, M. Hong, A. Takeda, P.K. Jawanpuria, B. Mishra, SLTrain: a sparse plus low rank approach for parameter and memory efficient pretraining, in: Advances in Neural Information Processing Systems,37, 2024, 118267–118295.

[31] S. Liu, X. Zhou, H. Chen, Multiscale temporal dynamic learning for time series classification, IEEE Trans. Knowl. Data Eng. 37 (2025) 3543–3555.

[32] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett, et al., H2o: heavy-hitter oracle for efficient generative inference of large language models, in: Advances in Neural Information Processing Systems,36, 2023, 34661–34710.

[33] Y. Li, Y. Huang, B. Yang, B. Venkitesh, A. Locatelli, H. Ye, T. Cai, P. Lewis, D. Chen, SnapKV: LLM knows what you are looking for before generation, (2024). arXiv preprint arXiv:2404.14469

[34] Y. Li, B. Dong, F. Guerin, C. Lin, Compressing context to enhance inference efficiency of large language models, in: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, 2023, pp. 6342–6353.

[35] H. Jiang, Q. Wu, X. Luo, D. Li, C.-Y. Lin, Y. Yang, L. Qiu, LongLLMLingua: accelerating and enhancing LLMs in long context scenarios via prompt compression, in: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2024, pp. 1658–1677.

[36] N. Kitaev, L. Kaiser, A. Levskaya, Reformer: the efficient transformer, in: International Conference on Learning Representations, 2019.

[37] A. Katharopoulos, A. Vyas, N. Pappas, F. Fleuret, Transformers are RNNS: fast autoregressive transformers with linear attention, in: International Conference on Machine Learning, PMLR, 2020, pp. 5156–5165.

[38] A. Gu, K. Goel, C. Re, Efficiently modeling long sequences with structured state spaces, in: International Conference on Learning Representations, 2022.

[39] A. Gu, T. Dao, Mamba: linear-time sequence modeling with selective state spaces, arXiv preprint arXiv:2312.00752 (2023).

[40] S.A. Jacobs, M. Tanaka, C. Zhang, M. Zhang, S.L. Song, S. Rajbhandari, Y. He, Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models, (2023). arXiv preprint arXiv:2309.14509

[41] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen, P. Liang, C. Ré, I. Stoica, C. Zhang, Flexgen: high-throughput generative inference of large language models with a single gpu, in: International Conference on Machine Learning, PMLR, 2023, pp. 31094–31116.

[42] Z. Dong, T. Tang, L. Li, W.X. Zhao, A survey on long text modeling with transformers, (2023). arXiv preprint arXiv:2302.14502

[43] R. Child, S. Gray, A. Radford, I. Sutskever, Generating long sequences with sparse transformers, (2019). arXiv preprint arXiv:1904.10509

[44] G. Zhao, J. Lin, Z. Zhang, X. Ren, Q. Su, X. Sun, Explicit sparse transformer: concentrated attention through explicit selection, (2019). arXiv preprint arXiv:1912.11637

[45] M. Zaheer, G. Guruganesh, K.A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, et al., Big bird: transformers for longer sequences, in: Advances in Neural Information Processing Systems,33, 2020, 17283–17297.

[46] I. Beltagy, M.E. Peters, A. Cohan, Longformer: the long-document transformer, (2020). arXiv preprint arXiv:2004.05150

[47] J. Ainslie, S. Ontanon, C. Alberti, V. Cvicek, Z. Fisher, P. Pham, A. Ravula, S. Sanghai, Q. Wang, L. Yang, ETC: encoding long and structured inputs in transformers, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020, pp. 268–284.

[48] S. Ge, Y. Zhang, L. Liu, M. Zhang, J. Han, J. Gao, Model tells you what to discard: adaptive KV cache compression for LLMs, in: The Twelfth International Conference on Learning Representations, 2024.

[49] G. Xiao, Y. Tian, B. Chen, S. Han, M. Lewis, Efficient streaming language models with attention sinks, in: The Twelfth International Conference on Learning Representations, 2024.

[50] C. Han, Q. Wang, H. Peng, W. Xiong, Y. Chen, H. Ji, S. Wang, LM-infinite: zero-shot extreme length generalization for large language models, in: Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), 2024, pp. 3991–4008.

[51] Z. Liu, A. Desai, F. Liao, W. Wang, V. Xie, Z. Xu, A. Kyrillidis, A. Shrivastava, Scissorhands: exploiting the persistence of importance hypothesis for LLM KV cache compression at test time, in: Advances in Neural Information Processing Systems, Adv. Neural Inf. Process. Syst. 36, 2024, 52342–52364.

[52] L. Ribar, I. Chelombiev, L. Hudlass-Galley, C. Blake, C. Luschi, D. Orr, SparQ attention: bandwidth-efficient LLM inference, in: Forty-first International Conference on Machine Learning, 2024.

[53] J. Dai, Z. Huang, H. Jiang, C. Chen, D. Cai, W. Bi, S. Shi, Sequence can secretly tell you what to discard, (2024). arXiv preprint arXiv:2404.15949

[54] J. Ren, S. Rajbhandari, R.Y. Aminabadi, O. Ruwase, S. Yang, M. Zhang, D. Li, Y. He, Zero-offload: democratizing {billion-scale} model training, in: 2021 USENIX Annual Technical Conference (USENIX ATC 21), 2021, pp. 551–564.

[55] S. Rajbhandari, J. Rasley, O. Ruwase, Y. He, Zero: memory optimizations toward training trillion parameter models, in: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2020, pp. 1–16.
[56] P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar, et al., Holistic evaluation of language models, (2022). arXiv preprint arXiv:2211.09110
[57] S. Narayan, S.B. Cohen, M. Lapata, Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization, (2018). arXiv preprint arXiv:1808.08745
[58] A. Narayan, I. Chami, L. Orr, S. Arora, C. Ré, Can foundation models wrangle your data?, (2022). arXiv preprint arXiv:2205.09911
[59] X. Chen, P. Maniatis, R. Singh, C. Sutton, H. Dai, M. Lin, D. Zhou, Spreadsheetcoder: formula prediction from semi-structured context, in: International Conference on Machine Learning, PMLR, 2021, pp. 1661–1672.
[60] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebron, S. Sanghai, GQA: training generalized multi-query transformer models from multi-head checkpoints, in: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, 2023, pp. 4895–4901.
[61] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, and others, Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems,32, 2019, 8026–8037.
[62] T. Dao, D. Fu, S. Ermon, A. Rudra, C. Ré, Flashattention: fast and memory-efficient exact attention with io-awareness, in: Advances in Neural Information Processing Systems,35, 2022, 16344–16359.
[63] L. Pekelis, M. Feil, F. Moret, M. Huang, T. Peng, Llama 3 gradient: a series of long context models, 2024. https://doi.org/10.57967/hf/3372
[64] V. Karpukhin, B. Oguz, S. Min, P.S.H. Lewis, L. Wu, S. Edunov, D. Chen, W.-t. Yih, Dense passage retrieval for open-domain question answering, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, 2020, pp. 6769–6781.
[65] X. Zhang, Y. Chen, S. Hu, Z. Xu, J. Chen, M. Hao, X. Han, Z. Thai, S. Wang, Z. Liu, M. Sun, ∞bench: extending long context evaluation beyond 100K tokens, in: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Bangkok, Thailand, 2024, pp. 15262–15277. https://aclanthology.org/2024.acl-long.15262.