# CompactAttention: Accelerating Chunked Prefill with Block-Union KV Selection

**Jiwon Song** **Dongwon Jo** **Beomseok Kang** **Jae-Joon Kim***
Seoul National University
{jiwon.song, dongwonjo, beomseok, kimjaejoon}@snu.ac.kr
https://github.com/jiwonsong-dev/CompactAttention

## Abstract

Chunked prefill has become a widely adopted serving strategy for long-context large language models, but efficient attention computation in this regime remains challenging. Existing sparse attention methods are primarily designed for one-shot prefill and do not translate efficiently to chunked prefill: block-sparse kernels lose efficiency when the query length is limited by the chunk size, while fine-grained pattern search becomes costly when repeated over the accumulated KV cache at every chunk. QUOKA, a recent method that directly targets chunked prefill, avoids sparse-kernel overhead but relies on query-subsampled, token-level KV selection, which can miss query-specific KV entries and introduce explicit KV-copy overhead. To address these limitations, we propose **CompactAttention**, a chunked-prefill attention mechanism based on *Block-Union KV Selection*. CompactAttention treats 2D block-sparse masks as KV-selection signals rather than direct sparse-kernel execution plans, and converts them into GQA-aware per-group KV block tables through Q-block union and intra-group union. This construction produces block tables that preserve all KV blocks selected by the input masks under paged execution constraints, enabling selected KV blocks to be accessed in place without explicit KV compaction. On LLaMA-3.1-8B-Instruct, CompactAttention maintains accuracy close to dense attention on the RULER benchmark while delivering up to 2.79× attention speedup over dense attention and 1.25× over FlashPrefill at 128K context length under chunked prefill.

## 1 Introduction

As large language models (LLMs) are increasingly used for long-horizon reasoning, document understanding, code analysis, and agentic workloads, their supported context windows have grown rapidly, reaching hundreds of thousands to even millions of tokens in recent proprietary and open-source models [1, 2, 3, 4, 5]. Processing such long contexts in a single prefill pass is increasingly impractical. First, full-sequence attention incurs quadratic compute cost with respect to context length, making one-shot prefill expensive at long contexts. Second, in online serving systems where prefill and decode requests are batched together, a long prefill pass can stall decode requests, making it difficult to satisfy time-between-token (TBT) service-level objectives (SLOs).

Chunked prefill [6, 7], now adopted in major serving frameworks such as vLLM [8] and SGLang [9], addresses these issues by processing long inputs sequentially in fixed-size chunks, each attending to both its own KVs and the accumulated KV cache from previous chunks. This makes efficient attention under chunked prefill an increasingly important problem.

The dominant approach to accelerating long-context prefill is block-sparse attention. Since FlashAttention [10, 11, 12] operates on blocks of tokens, block-sparse methods [13, 14, 15, 16, 17] first

*Corresponding Author

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

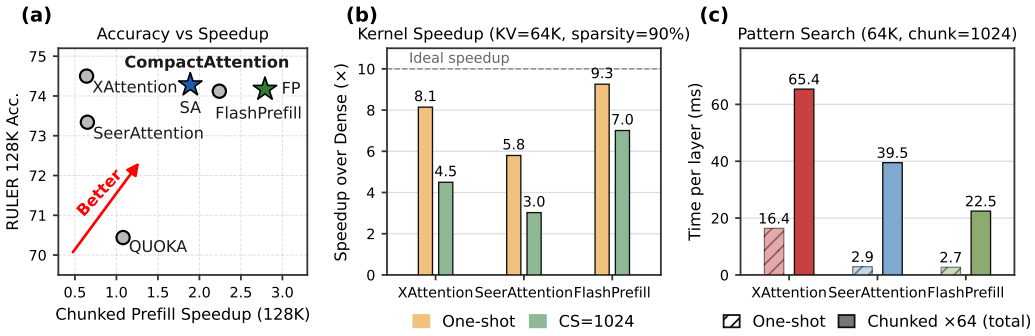



Figure 1: **(a)** CompactAttention achieves the best accuracy–speedup trade-off. **(b)** Block-sparse kernels under chunked prefill ($Q \ll KV$) fall far below one-shot and ideal speedups. **(c)** Pattern search cost accumulates across chunks, with XAttention incurring the highest overhead.

estimate which attention blocks are important and then compute only the selected subset of the attention map. These methods can be effective for one-shot prefill, where the query and key-value lengths are both large enough for sparse execution to amortize irregular memory-access overheads.

However, directly applying block-sparse attention to chunked prefill exposes two limitations. First, sparse execution becomes inefficient when $Q \ll KV$: block-sparse kernels have too few query blocks to expose sufficient parallelism and amortize irregular access overheads, so the achieved speedup falls far below what the nominal sparsity would suggest. Second, sparse pattern search must be repeated at every chunk over the accumulated KV cache, making cumulative search overhead a first-order concern and leaving only lightweight pattern-search mechanisms practical. An alternative is to perform dense attention over a selected subset of KV entries, avoiding block-sparse kernel overhead entirely.

QUOKA [18] is a representative method that directly targets chunked prefill by avoiding sparse kernels and performing dense attention over a reduced set of KV entries selected by a subsampled set of queries. However, it introduces two limitations. First, KV entries critical to non-sampled queries can be missed, leading to accuracy degradation on tasks requiring distributed information access. Second, token-level selection requires explicit KV gathering before attention execution, introducing copy overhead that grows with context length and batch size.

We propose **CompactAttention**, a chunked-prefill attention mechanism that decouples block-level KV selection from sparse-kernel execution. The key idea is to separate how KV blocks are selected from how they are executed: CompactAttention reuses lightweight block-sparse pattern search methods for selection, while lowering their 2D masks into GQA-aware KV block tables for zero-copy paged execution. It converts per-query-block, per-head masks into per-group KV block tables through Q-block union and intra-group union, producing tables that retain all selected KV blocks under paged execution constraints. These block tables are then passed to a paged attention kernel, which accesses the selected KV blocks in place without explicit KV compaction. By executing block-level KV selection through a Grouped-Query Attention [19] (GQA)-aware dense paged-attention backend, CompactAttention avoids both the kernel inefficiency of block-sparse attention and the copy overhead of token-level KV selection.

We evaluate CompactAttention on long-context LLMs under chunked prefill. As summarized in Figure 1(a), CompactAttention achieves the best accuracy–speedup trade-off among all baselines on the RULER benchmark, maintaining accuracy close to dense attention while delivering up to 2.79× speedup at 128K context length on H200. These results show that block-union KV table construction and zero-copy paged execution directly address the execution bottlenecks of existing chunked-prefill attention methods.

## 2 Motivation

### 2.1 Block Sparse Attention under Chunked Prefill

Block-sparse attention has been the dominant paradigm for accelerating long-context prefill. These methods identify input-dependent sparse patterns for each attention head and compute only a selected

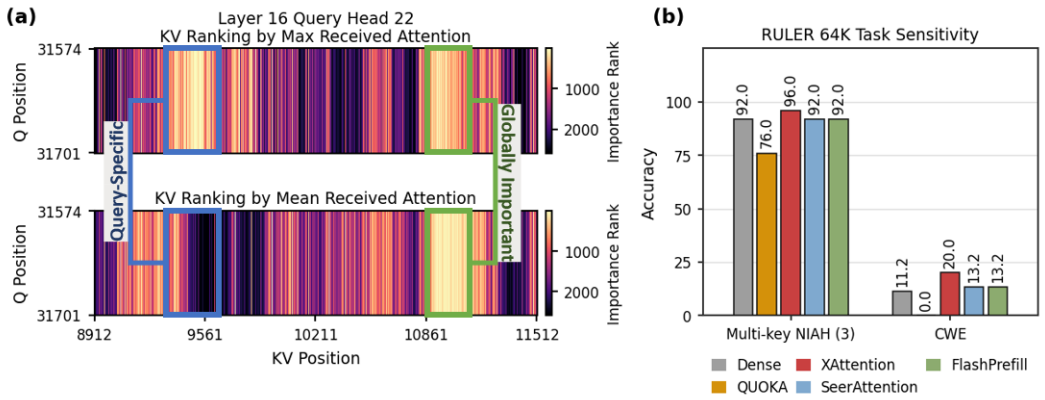



Figure 2: **(a)** KV-position rankings obtained by aggregating the attention each KV position receives from query positions in the shown window. Mean received attention ranks KV positions by their average received attention across all queries in the shown window, emphasizing globally important KV positions. Max received attention ranks KV positions by the largest attention received from any query within the window, exposing query-specific KV positions that may be important to only a few queries. Brighter colors indicate higher importance; highlighted regions show query-specific and globally important KV positions. **(b)** QUOKA degrades on tasks requiring distributed information access, while block-sparse methods remain close to dense attention.

subset of attention tiles, skipping tiles that are predicted to be unimportant. By exploiting sparsity while preserving most of the relevant attention computation, they can achieve substantial speedups with accuracy close to dense attention in one-shot prefill. However, applying these methods directly to chunked prefill exposes two key limitations.

**Kernel Inefficiency.** In chunked prefill, the query length at each iteration is limited to the chunk size, typically a few hundred to a thousand tokens in multi-request serving batches, while the key-value length grows as chunks accumulate. This $Q \ll KV$ regime differs substantially from one-shot prefill, where $Q = KV$. As shown in Figure 1(b), at the same KV length of 64K and 90% sparsity, block-sparse kernels achieve speedup much closer to the ideal ($10\times$) value under one-shot prefill ($Q = 64K$) than under chunked prefill ($Q = 1024$). This gap arises because block-sparse kernels rely on sufficiently large attention tiles to amortize the fixed overhead of sparse mask interpretation and irregular memory access. When the query sequence is short, the number of active query blocks is small, and this overhead dominates over the savings from skipping attention tiles.

**Pattern Search Overhead.** Another challenge is the cost of finding input-dependent sparse patterns. Reducing this cost has been a central focus of block-sparse attention research. For example, XAttention [15] substantially reduces scoring overhead compared with earlier fine-grained methods such as MInference [13] and FlexPrefill [14]. However, as shown in Figure 1(c), chunked prefill amplifies the cost of any online pattern search because scoring must be repeated at every chunk over the accumulated KV cache. This makes chunked prefill sensitive to the choice of pattern search method. Among existing block-sparse methods, lightweight selectors such as SeerAttention [16] and FlashPrefill [17] are therefore the most practical choices for this regime, although their cumulative search overhead remains higher under chunked prefill than under one-shot prefill. This constraint motivates a selector-agnostic execution design that can use practical lightweight methods today while remaining compatible with faster search mechanisms in the future.

### 2.2 Limitations of Query-Subsampled Direct KV Selection

Rather than using a sparse attention kernel, QUOKA [18] subsamples a subset of query tokens from the current chunk to score the importance of cached KV entries, and performs dense attention over the selected KV tokens. However, query-subsampled selection has an inherent coverage limitation. As illustrated in Figure 2(a), we rank each KV position by aggregating the attention it receives from query positions in the shown window. Mean-attention ranking highlights globally important KV positions that receive attention broadly across queries, while max-attention ranking reveals query-specific positions that receive strong attention from only a small subset of queries. Because

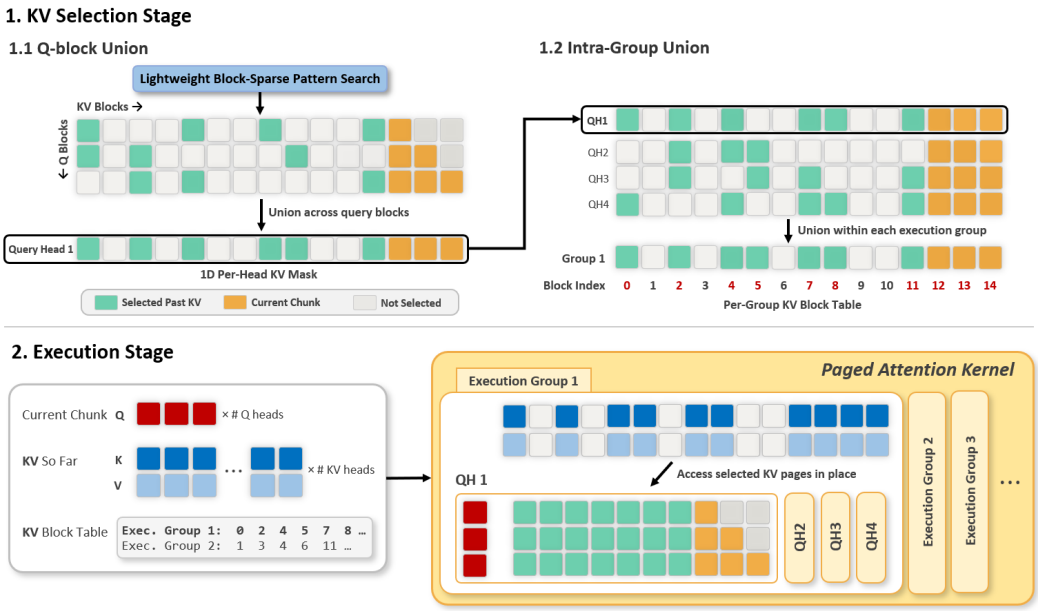



Figure 3: **Overview of CompactAttention.** The KV selection stage converts a 2D per-head block mask into per-group KV block tables through Q-block union and intra-group union. The execution stage passes these block tables to a paged attention kernel, which accesses selected KV pages in place without explicit KV compaction.

only sampled queries participate in QUOKA's KV scoring, such query-specific KV entries may be missed when their corresponding queries are not selected as evaluators. As shown in Figure 2(b), this coverage limitation appears on RULER tasks that require distributed information access. QUOKA degrades noticeably on Multi-key NIAH-3 and CWE, while block-sparse methods remain close to dense attention by evaluating all query blocks. Furthermore, unlike block-sparse methods that operate on contiguous blocks of tokens, QUOKA selects KV entries at token granularity. The selected KVs must therefore be gathered into a reduced KV tensor before attention execution, introducing explicit copy overhead that grows with context length and batch size.

These observations motivate a different design for chunked prefill. An effective mechanism should cover all query blocks, avoid sparse-kernel inefficiency in the short-query regime, and select KVs at block granularity for direct access without explicit compaction. **CompactAttention** is designed around these requirements by decoupling block-level KV selection from attention execution. This design is a trade-off rather than a free improvement: by keeping every KV block selected by any query block or head within an execution group, CompactAttention trades fine-grained sparsity for coverage of query-specific KV and for regular, copy-free execution.

# 3 CompactAttention

## 3.1 Overview

CompactAttention is a chunked prefill-aware attention mechanism that decouples sparse KV selection from execution, as illustrated in Figure 3. It can be combined with any lightweight block-sparse pattern search method that provides block-level importance estimates with low per-chunk overhead. Given block-level importance scores, CompactAttention proceeds in two stages: *selection* and *execution*.

In the *selection* stage, it converts per-head sparse masks into compact KV block tables through two union operations. First, it applies Q-block union across query blocks within the current chunk, which is necessary because dense paged attention consumes a single KV block list for the query blocks executed together rather than separate decisions for each ($Q$-block, $KV$-block) tile. Second, it applies intra-group union across query heads that are executed together, producing one KV block table

per group. In the *execution* stage, each per-group block table is passed to a paged attention kernel, which accesses the selected KV blocks in place without copying them into a separate buffer. This zero-copy execution avoids the copy overhead of token-level KV selection methods while leveraging optimized dense paged-attention kernels. The current chunk is always kept fully open to preserve causal attention semantics. Details of the *selection* process and the *execution* process are described in Section 3.2 and Section 3.3, respectively.

### 3.2 KV Selection: Block-Union KV Table Construction

CompactAttention's selection stage accepts any block-sparse pattern search method that produces a 2D per-head block mask with sufficiently low per-chunk overhead. In this work, we instantiate CompactAttention with two lightweight pattern search methods: SeerAttention (SA) [16], a learned attention-pattern predictor, and FlashPrefill (FP) [17], a training-free method based on max-threshold dynamic thresholding.

Let $M_{b,h,i,j} \in \{0,1\}$ denote the 2D block-sparse mask produced by the pattern search method for batch $b$, query head $h$, query block $i$, and KV block $j$. Existing block-sparse methods use this mask directly as a sparse-kernel execution plan. CompactAttention instead converts it into a KV block table that can be consumed by dense paged attention.

CompactAttention first applies *Q-block union* across query blocks:

$$\bar{M}_{b,h,j} = \bigvee_{i} M_{b,h,i,j}.$$

This produces a single 1D KV block mask per query head. The union is required because dense paged attention consumes one KV block list for the query blocks executed together. CompactAttention then applies *intra-group union* across query heads that share a KV block table:

$$G_{b,g,j} = \bigvee_{h \in \mathcal{H}(g)} \bar{M}_{b,h,j},$$

where $\mathcal{H}(g)$ denotes the set of query heads in an execution group $g$, which is a KV group by default. The resulting per-group page table is

$$T_{b,g} = \{j \mid G_{b,g,j} = 1\} = \{j \mid \exists h \in \mathcal{H}(g), \exists i,\ M_{b,h,i,j} = 1\}.$$

This block-union construction is coverage-preserving with respect to the input block-sparse mask: a KV block selected by any query block under any query head in the group remains selected in the resulting page table. Moreover, under the constraint that all query blocks and all query heads within an execution group share a single KV block table, $T_{b,g}$ is the minimal table that preserves this coverage. Any KV block outside $T_{b,g}$ is not selected by any query block or query head in the input mask, and can therefore be excluded without violating coverage preservation. Thus, the two union operations are not merely post-processing. They lower per-query-block, per-head sparse masks into GQA-aware paged KV tables, an executable representation for dense paged attention. This lowering preserves all KV blocks selected by the original 2D mask while enabling group-wise zero-copy execution.

The two union operations reduce sparsity compared to the original 2D block-sparse mask, because a KV block is retained if it is selected by any query block or query head in the execution group. However, we observe that this sparsity reduction can be compensated by using a more aggressive pattern search for the initial 2D mask while still preserving accuracy after union. As shown in Section 4.2, CompactAttention still achieves higher attention speedup than the corresponding block-sparse baselines, indicating that the execution advantage of dense paged attention outweighs the sparsity loss in practice.

For models with large GQA groups, applying intra-group union across the full group can cause excessive sparsity loss. We therefore split each KV group into smaller execution groups and apply intra-group union independently within each group. In our implementation, we use a subgroup size of four query heads as a conservative default; further details are provided in Appendix B.1.

### 3.3 Execution: Zero-Copy Paged Attention

CompactAttention executes the selected KV blocks using a paged dense-attention backend while avoiding explicit K/V compaction. The key requirement is to expose each selected KV block as a

page that the backend can access directly, even when different groups use different block tables. Thus, the block-union table produced in Section 3.2 must be represented as metadata over the original KV cache rather than as a newly materialized compact KV tensor.

As illustrated in Figure 4, a sequence-major KV cache layout $[B, L, H_{kv}, D]$ forces all KV heads to share the same block table. This is insufficient for CompactAttention because its block tables are group-dependent. CompactAttention therefore stores the accumulated KV cache in a KV-head-major layout, $[B, H_{kv}, L, D]$, where each (batch, KV head, block) triple corresponds to a contiguous [block size, $D$] memory region.

This layout is not merely an implementation detail: it turns selected KV blocks into metadata-addressable pages. CompactAttention constructs a ragged page list independently for each (batch, group) row, passing only metadata—`kv_indptr` and `kv_indices`—to the paged attention kernel while reusing the original K/V payloads in place. Further implementation details are provided in Appendix B.2.

This zero-copy design avoids explicit compaction into a newly allocated dense buffer, whose memory bandwidth overhead grows with context length, batch size, and the number of selected KV blocks. Since CompactAttention uses a standard paged dense-attention backend, improvements to dense attention kernels can be adopted without changing the selection stage.

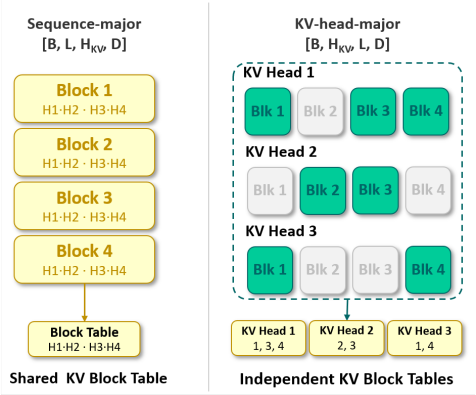



Figure 4: **KV cache layout comparison.** Sequence-major layout forces KV heads to share one block table, preventing independent block selection. KV-head-major layout exposes each KV-head block as a page, enabling independent KV block tables without copying K/V payloads.

# 4 Experiments

## 4.1 Experimental Setup

**Models.** We evaluate on two open-source models. LLaMA-3.1-8B-Instruct [20] is a dense LLM with a 128K-token context window. Qwen3-30B-A3B-Instruct-2507 [21] is a Mixture-of-Experts LLM with a 256K-token context window. Both models use Grouped-Query Attention (GQA). For accuracy evaluation, we use two long-context benchmarks: RULER [22] and LongBench V2 [23].

**Compared Methods.** We compare CompactAttention against several baselines. For dense attention, we use FlashInfer 0.6.9 [24] with FlashAttention-2 [11] and FlashAttention-3 [12] backends depending on the device. For block-sparse attention, we include SeerAttention [16] with block size 64, XAttention [15] with block size 128, and FlashPrefill [17] with block size 128. QUOKA [18] is the most directly comparable baseline for chunked-prefill KV selection; it selects KV entries via query subsampling and executes attention with a dense kernel.

CompactAttention uses the FlashInfer infrastructure as the paged attention execution backend, but supplies per-group KV block tables as page metadata to attend only to the selected KV blocks. CompactAttention-SA uses the pre-trained SeerAttention gate released for LLaMA-3.1-8B-Instruct without modification. CompactAttention-FP applies FlashPrefill's training-free thresholding and requires no model-specific adaptation, enabling evaluation on both models. In both cases, CompactAttention adopts the block size of the corresponding block-sparse attention method. As no pre-trained SeerAttention gate is available for Qwen3-30B-A3B-Instruct-2507, SeerAttention and CompactAttention-SA are evaluated only on LLaMA-3.1-8B-Instruct.

For QUOKA, we use the fixed 25% KV budget from the original paper. For all other sparse methods, we set sparsity hyperparameters independently for each method and model to select accuracy-preserving operating points. For XAttention, we construct a head-wise threshold table for each evaluated model using the official implementation. For LLaMA-3.1-8B-Instruct, we use threshold $= 3\text{e}{-4}$ for SeerAttention, threshold $= 5\text{e}{-4}$ for CompactAttention-SA, $\alpha = 0.01$ for FlashPrefill,

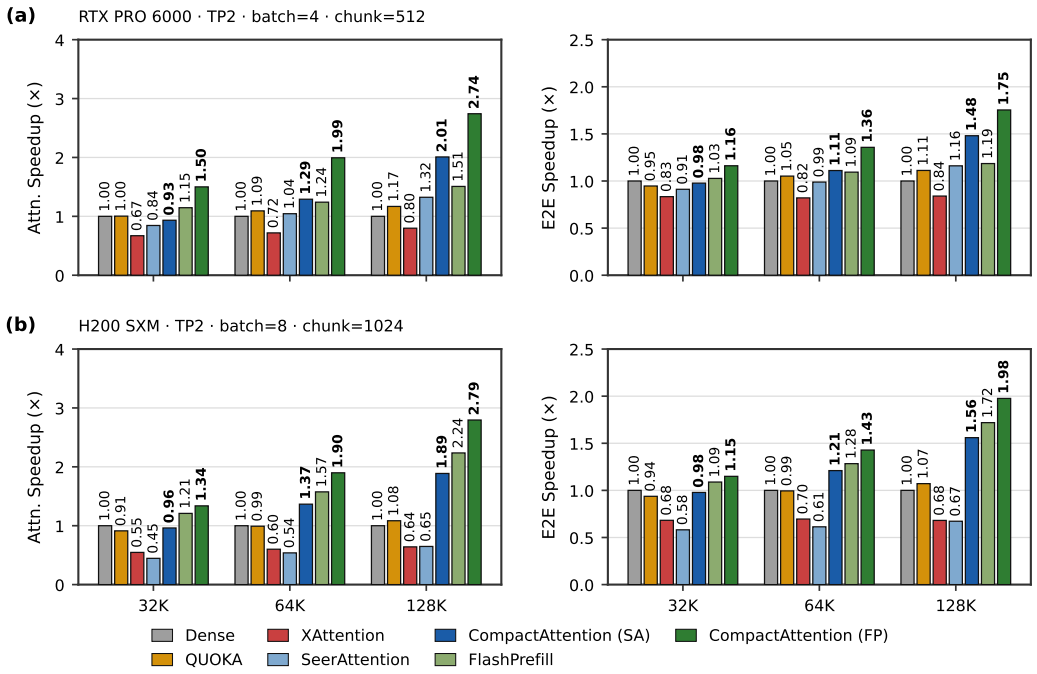



Figure 5: **Attention and end-to-end speedup under chunked prefill.** We report speedup over dense attention on LLaMA-3.1-8B-Instruct across context lengths. **(a)** RTX PRO 6000 GPUs with TP=2, batch size 4, and chunk size 512. **(b)** H200 SXM GPUs with TP=2, batch size 8, and chunk size 1024. CompactAttention achieves the largest gains at long context lengths, and the attention-level improvements translate into end-to-end latency reductions.

and $\alpha = 0.06$ for CompactAttention-FP. For Qwen3-30B-A3B-Instruct-2507, we use $\alpha = 0.02$ for FlashPrefill and $\alpha = 0.12$ for CompactAttention-FP.

**Environment.** We measure attention latency on two NVIDIA GPUs. The RTX PRO 6000 features 96 GB of GDDR7 memory and is based on the Blackwell microarchitecture (SM120), supporting FlashAttention-2. The H200 SXM provides 141 GB of HBM3e memory and is based on the Hopper microarchitecture (SM90), enabling FlashAttention-3 with Hopper-specific optimizations.

### 4.2 Speedup

Figure 5 reports attention-level and end-to-end speedup over the dense-attention baseline on LLaMA-3.1-8B-Instruct under chunked prefill, where end-to-end latency measures total wall-clock time for chunked prefill. We evaluate RTX PRO 6000 (TP=2, batch size 4, chunk size 512) and H200 SXM (TP=2, batch size 8, chunk size 1024). Raw LLaMA latency values and additional Qwen3-30B-A3B-Instruct-2507 speedup results are provided in Appendix C.1.

QUOKA achieves only limited speedup at long context lengths, as the token-level gather-and-pack overhead offsets the gain from attending to fewer tokens. XAttention and SeerAttention are often slower than dense attention, reflecting repeated pattern search overhead and inefficient block-sparse execution in the $Q \ll KV$ regime. FlashPrefill is the strongest block-sparse baseline, benefiting from lightweight pattern search and optimized block-sparse execution.

CompactAttention-SA and CompactAttention-FP show increasing speedup as context length grows. On H200 at 128K, CompactAttention-FP reaches 2.79× attention speedup and 1.98× end-to-end speedup over the dense-attention baseline. Against FlashPrefill, the strongest baseline, the gain is more modest: 1.25× in attention latency and 1.15× end-to-end at 128K. Both CompactAttention variants improve over their corresponding block-sparse baselines at long context lengths, showing that dense paged execution of the unioned KV set outweighs the sparsity reduction from block union.

Table 1: **RULER accuracy across context lengths.** All methods are evaluated with chunk size 1024.

| Method | LLaMA-3.1-8B-Instruct | | | | Qwen3-30B-A3B-Instruct-2507 | | | |
|---|---|---|---|---|---|---|---|---|
| | 32K | 64K | 128K | Avg | 32K | 64K | 128K | Avg |
| Dense | 88.23 | 83.52 | 76.59 | 82.78 | 93.01 | 90.56 | 87.81 | 90.46 |
| QUOKA | 83.52 | 79.99 | 70.44 | 77.98 | 81.96 | 79.57 | 79.03 | 80.19 |
| XAttention | 89.16 | 83.27 | 74.50 | 82.31 | 92.29 | 88.92 | 88.13 | 89.78 |
| SeerAttention | 90.23 | 83.82 | 73.34 | 82.46 | | | – | |
| **CompactAttention**-SA | 88.92 | 84.12 | 74.28 | 82.44 | | | – | |
| FlashPrefill | 88.99 | 83.94 | 74.12 | 82.35 | 91.55 | 87.73 | 86.77 | 88.68 |
| **CompactAttention**-FP | 88.77 | 83.96 | 74.17 | 82.30 | 92.41 | 88.43 | 87.26 | 89.37 |

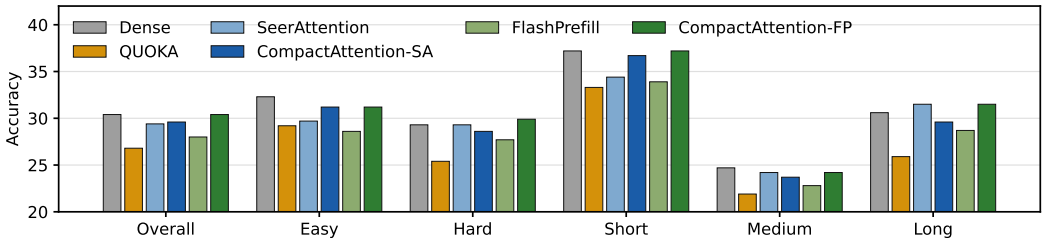



Figure 6: **LongBench V2 accuracy on LLaMA-3.1-8B-Instruct with chunk size 1024.** CompactAttention variants remain close to dense attention across difficulty levels and context-length groups, while QUOKA degrades more noticeably, especially on Hard samples.

### 4.3 Accuracy

**RULER.** Table 1 reports RULER accuracy across context lengths for all methods. QUOKA consistently underperforms dense attention across both models and context lengths. This is consistent with the coverage limitation discussed in Section 2.2: query-subsampled KV scoring can miss KV entries that are important to unsampled query positions, leading to degradation on tasks requiring distributed information access. Block-sparse attention methods—XAttention, SeerAttention, and FlashPrefill—remain close to dense attention, suggesting that block-level selectors that evaluate all query blocks better preserve the relevant attention computation.

CompactAttention-SA and CompactAttention-FP exhibit similar accuracy to their corresponding block-sparse baselines because they reuse the same block-level pattern search and preserve the selected blocks through Q-block union and intra-group union. Thus, CompactAttention largely preserves the selection quality of block-sparse methods while avoiding their sparse-kernel inefficiency under chunked prefill.

**LongBench V2.** Figure 6 reports LongBench V2 accuracy on LLaMA-3.1-8B-Instruct with chunk size 1024 across difficulty levels and context-length groups. The favorable accuracy trend of CompactAttention also extends to LongBench V2, which requires deeper understanding and reasoning over long contexts. Both CompactAttention variants remain close to dense attention, while QUOKA degrades noticeably, particularly on Hard samples. Although SeerAttention achieves comparable accuracy, Section 4.2 shows that CompactAttention provides a more favorable accuracy–efficiency trade-off under chunked prefill.

### 4.4 Ablation Studies

We focus ablations on CompactAttention-FP (CA-FP), the training-free instantiation applicable to both evaluated models.

**Sparsity Analysis.** Figure 7(a) compares the sparsity of FlashPrefill and CompactAttention-FP at the selected operating point on RULER 128K. FlashPrefill with $\alpha = 0.01$ achieves 71.8% sparsity. CompactAttention-FP (CA-FP) uses a more aggressive initial FlashPrefill-style mask with $\alpha = 0.06$, whose sparsity decreases from 90.2% to 69.7% after Q-block union and intra-group union. Thus, CompactAttention-FP reaches comparable executed sparsity to FlashPrefill while preserving the selected blocks required by the block-union table. Post-union keep ratios vary across layers and

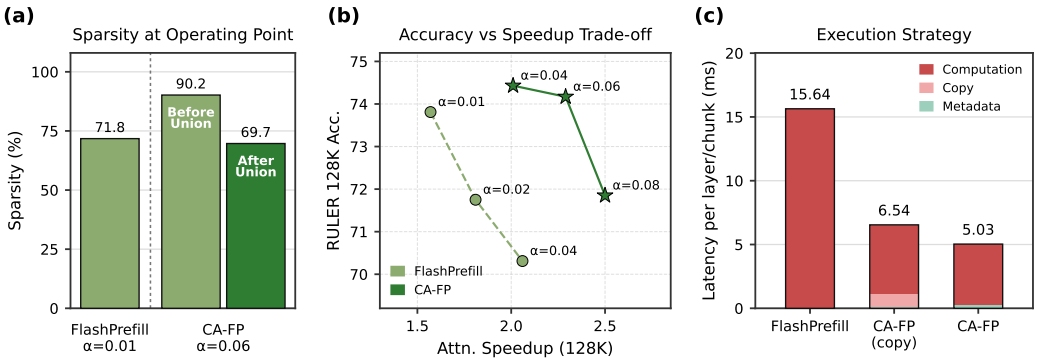



Figure 7: **(a)** Sparsity at the selected operating point. **(b)** Accuracy–speedup trade-off under $\alpha$ sweep on RULER 128K (RTX PRO 6000, TP=2, batch size 4, chunk size 1024). **(c)** Execution-only ablation at matched sparsity using the same unioned block mask (RTX PRO 6000, 128K, batch size 4, chunk size 512).

execution groups, and a few become nearly dense, but these account for a small share of the executed attention at long context; Appendix C.5 reports the per-stage, distributional, and cross-task analysis.

**Pattern-Search Aggressiveness.** Figure 7(b) evaluates the accuracy–speedup trade-off under different $\alpha$ values on RULER 128K. We sweep method-specific $\alpha$ values for FlashPrefill and CompactAttention-FP because block union changes the executed sparsity after the initial FlashPrefill-style mask is generated. CompactAttention-FP remains favorable in the high-accuracy operating region, achieving higher attention speedup than FlashPrefill at comparable or higher accuracy.

**Execution Strategy.** Figure 7(c) isolates the effect of execution strategy using the same unioned block mask at matched sparsity. The block-sparse variant executes this mask directly with a block-sparse kernel. CompactAttention-FP (Copy) gathers the selected KV blocks into a contiguous buffer before invoking dense attention, while CompactAttention-FP represents the same selected blocks as paged-attention metadata and accesses the original KV cache in place.

We separately report metadata overhead for each execution path. CompactAttention-FP incurs metadata overhead from constructing per-group block tables and planning paged attention execution, whereas CompactAttention-FP (Copy) incurs explicit KV-copy overhead from materializing selected blocks into a compact buffer. Since all variants use the same selected KV blocks, this ablation isolates the effect of execution. CompactAttention-FP achieves the lowest latency despite its metadata cost, showing that in-place paged execution is more efficient than either sparse-kernel execution or explicit KV compaction.

## 5 Limitations

CompactAttention inherits the quality of the underlying block-sparse pattern search method: blocks missed by the input mask cannot be recovered by block-union KV selection. It also trades sparsity for coverage, since Q-block union and intra-group union retain any KV block selected by any query block or query head within an execution group. As a result, the executed sparsity is not set by an explicit budget but is controlled indirectly through the selector hyperparameters and execution-group partitioning, and individual layers or execution groups can become nearly dense. While dense paged execution outweighs this sparsity loss in our evaluated settings, the trade-off may vary with model architecture, context length, chunk size, sparsity hyperparameters, and execution-group partitioning. CompactAttention is most effective when the accumulated KV cache is large enough to amortize pattern search and metadata construction overheads. Consistent with this behavior, our end-to-end latency gains become more pronounced as context length increases. Conversely, at short contexts or small batch sizes these overheads may not be amortized, and CompactAttention can be slower than dense attention or FlashPrefill. In addition, our implementation builds on HuggingFace Transformers and FlashInfer; integrating CompactAttention into serving frameworks such as vLLM or SGLang requires exposing per-execution-group block tables and a KV-head-major cache layout.

## 6 Conclusion

We presented CompactAttention, a chunked-prefill attention mechanism based on Block-Union KV Selection. CompactAttention treats 2D block-sparse masks as KV-selection signals rather than direct sparse-kernel execution plans, converts them into per-group KV block tables through Q-block union and intra-group union, and executes the selected KV blocks in place with zero-copy paged attention. Across RULER and LongBench V2, CompactAttention maintains accuracy close to dense attention while improving attention latency under chunked prefill, reaching up to 2.79× speedup over the dense-attention baseline and 1.25× over FlashPrefill at 128K context length on H200. These results demonstrate that the main bottleneck of sparse attention under chunked prefill is not only which KV blocks to select, but also how the selected blocks are executed. By making sparse selection compatible with efficient dense paged attention kernels, CompactAttention offers a practical path toward faster long-context LLM serving.

## Acknowledgments and Disclosure of Funding

This work was supported in part by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.RS-2026-25507427, Development of Efficient Architectures and Training Techniques for High-Performance Lightweight AI Models, No.RS-2025-02273157: Development of Low Power Training/Inference Technologies based on AI Semiconductors, RS-2023-00256081: artificial intelligence semiconductor support program to nurture the best talents, No. 2021-0-01343: Artificial Intelligence Graduate School Program (Seoul National University)), and BK21 FOUR program. (Corresponding Author: Jae-Joon Kim).

## References

[1] Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

[2] Anthropic. System card: Claude Opus 4.6. Technical report, Anthropic, February 2026. Accessed: 2026-04-29.

[3] Google DeepMind. Gemini 3 Pro model card. Technical report, Google DeepMind, November 2025. Model card update: December 2025. Accessed: 2026-04-29.

[4] Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, SH Cai, Yuan Cao, Y Charles, HS Che, Cheng Chen, Guanduo Chen, et al. Kimi k2. 5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

[5] DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

[6] Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gulavani, and Ramachandran Ramjee. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. *arXiv preprint arXiv:2308.16369*, 2023.

[7] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pages 117–134, 2024.

[8] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

[9] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

[10] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

[11] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[12] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *Advances in Neural Information Processing Systems*, 37:68658–68685, 2024.

[13] Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Advances in Neural Information Processing Systems*, 37:52481–52515, 2024.

[14] Xunhao Lai, Jianqiao Lu, Yao Luo, Yiyuan Ma, and Xun Zhou. Flexprefill: A context-aware sparse attention mechanism for efficient long-sequence inference. *arXiv preprint arXiv:2502.20766*, 2025.

[15] Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. Xattention: Block sparse attention with antidiagonal scoring. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

[16] Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden Kwok-Hay So, Ting Cao, Fan Yang, et al. Seerattention: Learning intrinsic sparse attention in your llms. *arXiv preprint arXiv:2410.13276*, 2024.

[17] Qihang Fan, Huaibo Huang, Zhiying Wu, Juqiu Wang, Bingning Wang, and Ran He. Flashprefill: Instantaneous pattern discovery and thresholding for ultra-fast long-context prefilling. *arXiv preprint arXiv:2603.06199*, 2026.

[18] Dalton Jones, Junyoung Park, Matthew Morse, Mingu Lee, Chris Lott, and Harper Langston. Quoka: Query-oriented kv selection for efficient llm prefill. *arXiv preprint arXiv:2602.08722*, 2026.

[19] Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit 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*, pages 4895–4901, 2023.

[20] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[21] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[22] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

[23] Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. *arXiv preprint arXiv:2412.15204*, 2024.

[24] Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. *Proceedings of Machine Learning and Systems*, 7, 2025.

[25] Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, et al. Moba: Mixture of block attention for long-context llms. *arXiv preprint arXiv:2502.13189*, 2025.

[26] Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23078–23097, 2025.

[27] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

[28] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: query-aware sparsity for efficient long-context llm inference. In *Proceedings of the 41st International Conference on Machine Learning*, pages 47901–47911, 2024.

# A Related Work

## A.1 Chunked Prefill

Chunked prefill was first proposed by Sarathi [6], which splits prefill requests into equal-sized chunks and interleaves them with decode iterations to improve GPU utilization and decode throughput. Sarathi-Serve [7] built on this idea to directly address the throughput-latency tradeoff, introducing stall-free scheduling that allows new requests to join a running batch without pausing ongoing decodes, simultaneously improving TBT latency and throughput. Chunked prefill has since been adopted as the default scheduling strategy in major serving frameworks including vLLM [8] and SGLang [9].

## A.2 Sparse Attention for Long-Context Prefill

A line of work accelerates prefill by skipping unimportant attention blocks. MInference [13] classifies each attention head's dominant pattern offline and applies the corresponding sparse computation. FlexPrefill [14] dynamically selects sparse patterns and block indices per head and input via online scoring. XAttention [15] proposes a lightweight scoring mechanism based on anti-diagonal attention values to reduce the overhead of block selection. FlashPrefill [17] further reduces pattern search overhead and achieves high sparsity through fused dynamic thresholding. Taking a learning-based approach, SeerAttention [16] trains a lightweight AttnGate via self-distillation to predict block-level attention pattern while keeping original model weights frozen, achieving high sparsity with low pattern search overhead. Going further, MoBA [25] and NSA [26] integrate sparsity directly into the model through pretraining or continued training.

These methods succeed in achieving substantial speedup on one-shot prefill without substantially compromising accuracy. However, none of them address the kernel inefficiency or per-chunk scoring overhead that arise specifically under chunked prefill.

## A.3 Query-Dependent KV Selection

A related line of work reduces the KV cache retained after prefill by selecting only the entries deemed important for decoding. SnapKV [27] identifies important KV entries using the most recent query tokens as evaluators, discarding the rest before decoding begins. Quest [28] dynamically selects relevant KV pages at the page level in a query-aware manner at every decoding step. QUOKA [18] adapts this idea to chunked prefill, sampling a representative subset of query tokens from each chunk as evaluators to score KV importance and performing dense attention over the selected entries. However, as discussed in Section 2, query subsampling introduces a structural coverage limitation and token-level selection incurs explicit copy overhead, motivating the block-level selection and zero-copy execution design of CompactAttention.

# B Implementation Details

## B.1 Sub-KV-Group Union

For models with GQA ratio greater than 4:1 (e.g., Qwen3-30B-A3B-Instruct-2507 with 8:1), applying intra-group union across the full KV group causes excessive sparsity loss. CompactAttention therefore partitions each KV group into subgroups of four query heads and treats each subgroup as an execution group. We refer to this variant as sub-KV-group union. We use a subgroup size of 4 as a conservative default rather than a tuned optimum. Smaller subgroups preserve more sparsity but increase the number of block tables to construct, so the best size depends on the context length and chunk size.

As shown in Table 2, sub-KV-group union substantially lowers the keep ratio compared with full KV-group union while maintaining comparable accuracy and the same zero-copy execution interface.

Table 2: Subgroup-size trade-off on Qwen3-30B-A3B-Instruct-2507 at 128K context length. Latency is measured on H200 SXM GPUs with TP=2 and batch size 4. Subgroup size 8 corresponds to full KV-group union.

| Chunk size | $\alpha$ | Subgroup size | Keep ratio | RULER 128K | Attention latency (ms) |
|---|---|---|---|---|---|
| 1024 | 0.08 | 2 | 0.267 | 86.58 | 16147.8 |
| | | 4 | 0.365 | 87.28 | 18771.1 |
| | | 8 | 0.468 | 87.22 | 21951.6 |
| | 0.12 | 2 | 0.200 | 87.03 | 14790.7 |
| | | 4 | 0.280 | 87.26 | 16514.6 |
| | | 8 | 0.372 | 87.60 | 19732.4 |
| | 0.16 | 2 | 0.160 | 87.02 | 14355.9 |
| | | 4 | 0.225 | 87.86 | 15597.4 |
| | | 8 | 0.306 | 87.76 | 18390.4 |
| 512 | 0.12 | 2 | 0.160 | 85.86 | 32084.0 |
| | | 4 | 0.229 | 86.37 | 31052.3 |
| | | 8 | 0.313 | 85.91 | 32260.9 |

## B.2 Zero-Copy Paged Execution

CompactAttention uses FlashInfer [24] 0.6.9 as the paged attention execution backend. To support group-dependent block tables without copying K/V payloads, the accumulated KV cache is stored in a KV-head-major layout $[B, H_{kv}, L, D]$. Each $(\text{batch}, \text{KV head}, \text{block})$ triple then corresponds to a contiguous $[\text{block size}, D]$ memory region, which can be directly reinterpreted as a page without data movement.

To construct ragged page lists, the block-union mask is converted into CSR-style metadata $(\texttt{kv_indptr}, \texttt{kv_indices})$ by a fused CUDA kernel. Only this metadata is passed to the paged attention backend; K/V payloads are never copied. In practice, we flatten the batch and KV-head dimensions into a pseudo-batch dimension and call the backend with $\texttt{num_kv_heads} = 1$, so that each pseudo-sequence carries its own page list. For sub-KV-group union, query heads within the same execution group share the corresponding page list, enabling independent block tables for different execution groups without copying K/V payloads.

The current chunk is always kept fully open in the block mask. If the current chunk were sparsified, causal masking would be applied in compacted-position space rather than original absolute-position space, breaking causal attention semantics.

# C Experiment Details

## C.1 Additional Latency Results

**LLaMA-3.1-8B-Instruct.** Tables 3 and 4 report raw attention and end-to-end latency measurements for LLaMA-3.1-8B-Instruct under chunked prefill. Table 3 uses RTX PRO 6000 GPUs with TP=2, batch size 4, and chunk size 512, while Table 4 uses H200 SXM GPUs with TP=2, batch size 8, and chunk size 1024.

Both CompactAttention variants reduce attention latency at long context lengths compared with their corresponding block-sparse baselines and dense attention, and these improvements also translate into end-to-end latency reductions. At shorter context lengths, sparse methods can be slower in attention latency because pattern search and metadata construction overheads are not yet amortized, but their impact on end-to-end latency is smaller because attention accounts for a smaller fraction of total prefill time.

Table 3: LLaMA-3.1-8B-Instruct attention and end-to-end latency (ms) across context lengths under chunked prefill with chunk size 512, measured on RTX PRO 6000 GPUs with TP=2 and batch size 4.

| Method | Context Length | | | | |
|---|---|---|---|---|---|
| | 8K | 16K | 32K | 64K | 128K |
| **Attention Latency** | | | | | |
| Dense | 423.1 | 1224.0 | 5213.3 | 17838.8 | 65971.1 |
| QUOKA | 674.0 | 1599.5 | 5196.5 | 16325.4 | 56475.3 |
| XAttention | 1095.5 | 2700.5 | 7770.5 | 24819.2 | 82617.0 |
| SeerAttention | 1012.3 | 2548.7 | 6171.8 | 17078.2 | 49839.8 |
| **CompactAttention**-SA | 1303.0 | 2675.7 | 5584.4 | 13823.5 | 32846.0 |
| FlashPrefill | 496.6 | 1356.2 | 4547.7 | 14381.8 | 43750.5 |
| **CompactAttention**-FP | 596.0 | 1393.6 | 3474.8 | 8945.9 | 24055.4 |
| **End-to-End Latency** | | | | | |
| Dense | 2299.7 | 4858.0 | 12421.8 | 32177.4 | 94944.3 |
| QUOKA | 2470.4 | 5156.3 | 13116.8 | 30588.0 | 85375.3 |
| XAttention | 2874.8 | 6243.9 | 14896.8 | 39206.6 | 113016.5 |
| SeerAttention | 2911.3 | 6286.4 | 13620.0 | 32545.8 | 81774.4 |
| **CompactAttention**-SA | 3139.8 | 6382.9 | 12706.7 | 28960.4 | 64126.7 |
| FlashPrefill | 2294.2 | 4978.7 | 12088.4 | 29411.7 | 80108.5 |
| **CompactAttention**-FP | 2345.1 | 4953.2 | 10691.7 | 23707.8 | 54130.6 |

Table 4: LLaMA-3.1-8B-Instruct attention and end-to-end latency (ms) across context lengths under chunked prefill with chunk size 1024, measured on H200 SXM GPUs with TP=2 and batch size 8.

| Method | Context Length | | | | |
|---|---|---|---|---|---|
| | 8K | 16K | 32K | 64K | 128K |
| **Attention Latency** | | | | | |
| Dense | 438.6 | 1294.7 | 4225.4 | 14882.5 | 55619.7 |
| QUOKA | 582.3 | 1516.6 | 4629.7 | 15012.2 | 51292.4 |
| XAttention | 923.9 | 2600.1 | 7729.7 | 24777.8 | 86821.4 |
| SeerAttention | 1002.5 | 3073.4 | 9478.1 | 27632.5 | 85852.0 |
| **CompactAttention**-SA | 780.2 | 1775.7 | 4393.8 | 10895.6 | 29468.1 |
| FlashPrefill | 474.1 | 1279.6 | 3494.6 | 9451.1 | 24876.1 |
| **CompactAttention**-FP | 533.6 | 1280.3 | 3160.1 | 7839.3 | 19899.9 |
| **End-to-End Latency** | | | | | |
| Dense | 1394.3 | 3160.7 | 7948.6 | 22327.7 | 70754.4 |
| QUOKA | 1580.0 | 3571.4 | 8482.5 | 22451.8 | 66029.1 |
| XAttention | 1877.0 | 4536.7 | 11649.8 | 32093.5 | 103870.2 |
| SeerAttention | 1991.7 | 5029.2 | 13671.0 | 36406.8 | 105207.6 |
| **CompactAttention**-SA | 1788.3 | 3745.1 | 8125.8 | 18453.3 | 45366.8 |
| FlashPrefill | 1397.4 | 3135.1 | 7302.6 | 17392.8 | 41161.7 |
| **CompactAttention**-FP | 1452.7 | 3128.3 | 6917.7 | 15634.6 | 35806.2 |

**Chunk Size Sensitivity.** Table 5 reports attention latency under different chunk sizes on LLaMA-3.1-8B-Instruct at 128K context length, measured on H200 SXM GPUs with TP=2 and batch size 8. As chunk size increases, the number of chunked-prefill iterations decreases, reducing total attention latency. CompactAttention-FP consistently improves latency across all chunk sizes. Its relative speedup decreases as the chunk size grows because larger chunks increase Q-block union within each chunk, reducing effective sparsity, but it remains substantially faster than dense attention.

Table 5: Chunk-size sensitivity of attention latency on LLaMA-3.1-8B-Instruct at 128K context length, measured on H200 SXM GPUs with TP=2 and batch size 8.

| Method | Chunk Size | | |
|---|---|---|---|
| | 512 | 1024 | 2048 |
| Dense | 72598.6 | 55619.7 | 47478.9 |
| **CompactAttention**-FP | 25076.0 | 19899.9 | 19173.8 |
| Speedup | 2.90× | 2.79× | 2.48× |

**Attention Speedup on Qwen3-30B-A3B.** Table 6 reports additional attention speedup results on Qwen3-30B-A3B-Instruct-2507, a larger MoE model. CompactAttention-FP uses sub-KV-group union with subgroup size 4, as described in Appendix B.1. The results show that CompactAttention-FP provides long-context latency gains as the accumulated KV cache grows: it is faster than FlashPrefill at all evaluated lengths and faster than dense attention from 64K onward.

Table 6: Attention latency speedup on Qwen3-30B-A3B-Instruct-2507 under chunked prefill, measured on H200 SXM GPUs with TP=2, batch size 4, and chunk size 1024. MoE experts use a fused grouped-GEMM implementation.

| Method | 32K | 64K | 128K |
|---|---|---|---|
| Dense | 1.00 | 1.00 | 1.00 |
| FlashPrefill | 0.82 | 0.99 | 1.24 |
| **CompactAttention**-FP | 0.96 | 1.31 | 2.05 |

**Crossover across Chunk and Batch Sizes.** Tables 7 and 8 report attention latency across chunk sizes and batch sizes. With batch size 8, CompactAttention-FP becomes faster than dense attention at 32K, 16K, and 16K for chunk sizes 512, 1024, and 2048, and faster than FlashPrefill at 32K, 32K, and 16K, respectively. With chunk size 1024, it does not overtake dense attention up to 128K at batch size 1, and overtakes it at 64K, 16K, and 16K for batch sizes 4, 8, and 16; against FlashPrefill, the crossover occurs at 64K, 64K, 32K, and 16K for batch sizes 1, 4, 8, and 16.

Table 7: LLaMA-3.1-8B-Instruct attention latency (ms) across chunk sizes and context lengths under chunked prefill, measured on H200 SXM GPUs with TP=2 and batch size 8 (mean ± std over five runs).

| Method | Context Length | | | | |
|---|---|---|---|---|---|
| | 8K | 16K | 32K | 64K | 128K |
| **Chunk Size 512** | | | | | |
| Dense | 564.3±10.4 | 1628.3±13.2 | 5349.4±46.9 | 19156.4±55.9 | 72598.6±230.6 |
| FlashPrefill | 843.4±32.9 | 1796.1±20.7 | 4621.6±13.9 | 11379.1±70.6 | 30534.7±283.9 |
| **CompactAttention**-FP | 1152.2±15.6 | 2181.8±37.5 | 4350.9±14.6 | 10473.0±43.6 | 25076.0±66.8 |
| **Chunk Size 1024** | | | | | |
| Dense | 438.6±1.6 | 1294.7±0.7 | 4225.4±11.0 | 14882.5±17.9 | 55619.7±34.0 |
| FlashPrefill | 474.1±1.2 | 1279.6±2.5 | 3494.6±4.4 | 9451.1±30.0 | 24876.1±71.6 |
| **CompactAttention**-FP | 533.6±3.2 | 1280.3±1.7 | 3160.1±45.9 | 7839.3±55.3 | 19899.9±48.0 |
| **Chunk Size 2048** | | | | | |
| Dense | 397.6±1.6 | 1145.7±5.3 | 3660.9±15.2 | 12740.2±19.4 | 47478.9±30.9 |
| FlashPrefill | 438.4±2.3 | 1194.5±3.1 | 3250.2±2.9 | 8650.9±30.1 | 22106.5±97.3 |
| **CompactAttention**-FP | 446.6±0.5 | 1139.3±3.5 | 2891.1±5.3 | 7376.0±41.0 | 19173.8±268.3 |

Table 8: LLaMA-3.1-8B-Instruct attention latency (ms) across batch sizes and context lengths under chunked prefill, measured on H200 SXM GPUs with TP=2 and chunk size 1024 (mean ± std over five runs).

| Method | Context Length | | | | |
|---|---|---|---|---|---|
| | 8K | 16K | 32K | 64K | 128K |
| **Batch Size 1** | | | | | |
| Dense | 342.2±3.9 | 626.3±72.9 | 1392.9±25.1 | 3160.5±11.5 | 7715.4±251.9 |
| FlashPrefill | 541.4±2.4 | 1063.7±55.7 | 2125.6±238.6 | 5022.2±166.2 | 10007.7±565.4 |
| **CompactAttention**-FP | 572.6±26.3 | 1196.9±49.1 | 2376.6±40.8 | 4782.9±43.6 | 9899.2±68.7 |
| **Batch Size 4** | | | | | |
| Dense | 250.1±4.7 | 668.5±6.7 | 2114.8±4.5 | 7483.9±19.4 | 27895.9±19.4 |
| FlashPrefill | 435.1±40.0 | 923.0±59.5 | 2149.1±3.2 | 5462.4±13.0 | 14012.8±120.9 |
| **CompactAttention**-FP | 533.0±32.9 | 1064.5±26.0 | 2190.4±87.9 | 4735.6±28.4 | 11737.8±73.2 |
| **Batch Size 8** | | | | | |
| Dense | 438.6±1.6 | 1294.7±0.7 | 4225.4±11.0 | 14882.5±17.9 | 55619.7±34.0 |
| FlashPrefill | 474.1±1.2 | 1279.6±2.5 | 3494.6±4.4 | 9451.1±30.0 | 24876.1±71.6 |
| **CompactAttention**-FP | 533.6±3.2 | 1280.3±1.7 | 3160.1±45.9 | 7839.3±55.3 | 19899.9±48.0 |
| **Batch Size 16** | | | | | |
| Dense | 875.2±1.8 | 2606.9±4.9 | 8501.6±35.2 | 30680.5±11.5 | OOM |
| FlashPrefill | 904.9±1.6 | 2461.1±8.0 | 6732.6±9.3 | 17994.9±11.5 | OOM |
| **CompactAttention**-FP | 949.8±3.1 | 2332.5±7.1 | 5857.7±34.5 | 14728.1±37.3 | OOM |

### C.2 Additional Accuracy Results

**RULER with Chunk Size 512.** Table 9 reports RULER accuracy with chunk size 512 on LLaMA-3.1-8B-Instruct. CompactAttention variants maintain accuracy close to dense attention across context lengths, with modest degradation at 128K.

Table 9: **RULER accuracy across context lengths with chunk size 512.** We report Dense and CompactAttention variants on LLaMA-3.1-8B-Instruct.

| Method | LLaMA-3.1-8B-Instruct | | | |
|---|---|---|---|---|
| | 32K | 64K | 128K | Avg |
| Dense | 88.23 | 83.52 | 76.59 | 82.78 |
| **CompactAttention**-SA | 89.24 | 83.47 | 74.56 | 82.42 |
| **CompactAttention**-FP | 88.26 | 83.59 | 73.48 | 81.78 |

**LongBench V2.** Table 10 reports the full LongBench V2 breakdown for all methods on LLaMA-3.1-8B-Instruct. As discussed in Section 4.3, QUOKA shows consistent degradation across categories compared to dense attention, while block-sparse methods largely preserve accuracy. Both CompactAttention-SA and CompactAttention-FP maintain accuracy comparable to dense attention and their corresponding block-sparse counterparts across all difficulty levels and context lengths.

Table 10: LLaMA-3.1-8B-Instruct LongBench V2 accuracy comparison across task categories with chunk size 1024.

| Method | Overall | Easy | Hard | Short | Medium | Long |
|---|---|---|---|---|---|---|
| Dense | 30.4 | 32.3 | 29.3 | 37.2 | 24.7 | 30.6 |
| QUOKA | 26.8 | 29.2 | 25.4 | 33.3 | 21.9 | 25.9 |
| XAttention | 28.2 | 30.2 | 27.0 | 35.6 | 24.2 | 24.1 |
| SeerAttention | 29.4 | 29.7 | 29.3 | 34.4 | 24.2 | 31.5 |
| **CompactAttention**-SA | 29.6 | 31.2 | 28.6 | 36.7 | 23.7 | 29.6 |
| FlashPrefill | 28.0 | 28.6 | 27.7 | 33.9 | 22.8 | 28.7 |
| **CompactAttention**-FP | 30.4 | 31.2 | 29.9 | 37.2 | 24.2 | 31.5 |

### C.3 Copy-Based vs. Zero-Copy Execution

Table 11 extends the execution-strategy ablation in Figure 7(c) across batch sizes. CompactAttention-FP (Copy) materializes selected KV blocks into a compact buffer and then invokes dense attention directly with FlashAttention-2. Its metadata overhead includes the preprocessing required to determine where selected KV blocks should be copied in the compact buffer. In contrast, CompactAttention-FP eliminates explicit KV copying and represents selected KV blocks as paged-attention metadata. Its metadata overhead includes per-execution-group block-table construction and the `plan()` overhead of the FlashInfer wrapper. The results show that copy overhead in CompactAttention-FP (Copy) grows with batch size, whereas the metadata overhead of CompactAttention-FP grows much more slowly, supporting the benefit of zero-copy paged execution in batched chunked prefill.

Table 11: Batch-size scaling of copy and metadata overhead for CompactAttention-FP. Latency is measured in milliseconds.

| Batch Size | **CompactAttention**-FP (Copy) | | | **CompactAttention**-FP | |
|---|---|---|---|---|---|
| | Metadata | Copy | Computation | Metadata | Computation |
| 1 | 0.20 | 0.30 | 1.65 | 0.29 | 1.54 |
| 2 | 0.13 | 0.52 | 2.79 | 0.24 | 2.96 |
| 4 | 0.13 | 1.01 | 5.13 | 0.31 | 4.48 |
| 8 | 0.13 | 1.99 | 9.59 | 0.47 | 8.33 |
| 16 | 0.16 | 3.97 | 21.26 | 0.89 | 15.73 |

Table 12 compares the two execution strategies at the attention level on H200: with the same selected KV blocks, CompactAttention-FP (Copy) already achieves 2.62× speedup over dense attention at

128K, and zero-copy execution raises this to 2.79×, showing that most of the gain comes from reducing the executed KV set while zero-copy removes the remaining copy cost.

Table 12: Attention latency (ms) of copy-based and zero-copy execution with the same selected KV blocks on LLaMA-3.1-8B-Instruct, measured on H200 SXM GPUs with TP=2, batch size 8, and chunk size 1024.

| Method | Context Length | | |
|---|---|---|---|
| | 32K | 64K | 128K |
| Dense | 4225.4 | 14882.5 | 55619.7 |
| FlashPrefill | 3494.6 | 9451.1 | 24876.1 |
| **CompactAttention**-FP (Copy) | 3320.6 | 8223.6 | 21189.6 |
| **CompactAttention**-FP | 3160.1 | 7839.3 | 19899.9 |

### C.4 Comparison with FlexPrefill

We evaluate FlexPrefill [14] as an additional baseline under chunked prefill. Since the official implementation supports only batch size 1 and equal query and key lengths, we extend it to the rectangular attention of chunked prefill without changing its pattern search, and use the hyperparameters from the original paper. As shown in Table 13, FlexPrefill maintains competitive accuracy but is 2.99–4.81× slower than dense attention, as its fine-grained pattern search is repeated at every chunk over the growing KV cache.

Table 13: Comparison with FlexPrefill on LLaMA-3.1-8B-Instruct (chunk size 1024). Latency is measured on H200 with TP=2 and batch size 8.

| Method | RULER Accuracy | | | Attention Latency (ms) | | |
|---|---|---|---|---|---|---|
| | 32K | 64K | 128K | 32K | 64K | 128K |
| Dense | 88.23 | 83.52 | 76.59 | 4225.4 | 14882.5 | 55619.7 |
| FlexPrefill | 89.42 | 82.96 | 73.30 | 20339.1 | 56731.9 | 166356.5 |
| **CompactAttention**-FP | 88.77 | 83.96 | 74.17 | 3160.1 | 7839.3 | 19899.9 |

### C.5 Post-Union Sparsity Analysis

Keep ratios in this section are compute-weighted: kept and total KV blocks are summed over chunk iterations before taking the ratio, counting mandatory recent blocks, including the current chunk, in both. Medians, P90 values, and the fraction of near-dense executions (keep ratio $\geq 0.9$) are weighted by KV length over individual (chunk, layer, execution group) executions.

Table 14 decomposes the keep ratio of CompactAttention-FP by aggregation step. Q-block union and intra-group union each increase the keep ratio, and larger chunks increase it further. Near-dense executions become rare as the context grows, and CompactAttention-FP remains faster than both dense attention and FlashPrefill in all settings. Table 15 shows that the post-union keep ratio is similar across RULER tasks. Table 16 shows that on Qwen3-30B-A3B the keep ratio varies widely across layers and execution groups, and that sub-KV-group union keeps it lower than full KV-group union.

Table 14: Compute-weighted keep ratio after each aggregation step and attention speedup of CompactAttention-FP on LLaMA-3.1-8B-Instruct (RULER VT). Speedup is measured on H200 with TP=2 and batch size 8.

| Context | Chunk | Pre-union | Q-block union | Group union | | | | Speedup over | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | Mean | Median | P90 | $\geq 0.9$ | Dense | FlashPrefill |
| 32K | 512 | 0.281 | 0.406 | 0.584 | 0.538 | 0.994 | 0.206 | 1.23× | 1.06× |
| | 1024 | 0.274 | 0.477 | 0.652 | 0.634 | 1.000 | 0.277 | 1.34× | 1.11× |
| | 2048 | 0.260 | 0.552 | 0.717 | 0.732 | 1.000 | 0.352 | 1.27× | 1.12× |
| 64K | 512 | 0.179 | 0.260 | 0.415 | 0.333 | 0.887 | 0.095 | 1.83× | 1.09× |
| | 1024 | 0.174 | 0.308 | 0.468 | 0.397 | 0.949 | 0.125 | 1.90× | 1.21× |
| | 2048 | 0.168 | 0.364 | 0.525 | 0.467 | 0.993 | 0.159 | 1.73× | 1.17× |
| 128K | 512 | 0.100 | 0.150 | 0.260 | 0.184 | 0.589 | 0.035 | 2.90× | 1.22× |
| | 1024 | 0.098 | 0.182 | 0.303 | 0.227 | 0.671 | 0.049 | 2.79× | 1.25× |
| | 2048 | 0.096 | 0.223 | 0.354 | 0.281 | 0.754 | 0.064 | 2.48× | 1.15× |

Table 15: Compute-weighted post-union keep ratio of CompactAttention-FP across RULER tasks on LLaMA-3.1-8B-Instruct (chunk size 1024).

| Context | VT | NIAH-1 | CWE | QA-1 |
|---|---|---|---|---|
| 32K | 0.652 | 0.675 | 0.651 | 0.699 |
| 64K | 0.468 | 0.469 | 0.422 | 0.494 |
| 128K | 0.303 | 0.299 | 0.269 | 0.351 |

Table 16: Compute-weighted post-union keep ratio per execution group on Qwen3-30B-A3B-Instruct-2507 (128K, chunk size 1024) for the three layers with the highest and lowest average keep ratio under subgroup size 4. The last column reports the layer average under full KV-group union (subgroup size 8).

| Layer | Subgroup size 4 | | | | | | | | | Subgroup size 8 |
|---|---|---|---|---|---|---|---|---|---|---|
| | G0 | G1 | G2 | G3 | G4 | G5 | G6 | G7 | Avg. | Avg. |
| 2 | 0.654 | 0.438 | 0.303 | 0.367 | 0.873 | 0.428 | 0.505 | 0.508 | 0.510 | 0.636 |
| 0 | 0.758 | 0.973 | 0.324 | 0.393 | 0.626 | 0.311 | 0.206 | 0.081 | 0.459 | 0.598 |
| 38 | 0.319 | 0.534 | 0.574 | 0.466 | 0.465 | 0.161 | 0.291 | 0.466 | 0.410 | 0.490 |
| 7 | 0.218 | 0.166 | 0.123 | 0.133 | 0.165 | 0.123 | 0.097 | 0.096 | 0.140 | 0.159 |
| 5 | 0.105 | 0.177 | 0.168 | 0.115 | 0.120 | 0.145 | 0.080 | 0.159 | 0.134 | 0.161 |
| 8 | 0.051 | 0.065 | 0.029 | 0.170 | 0.129 | 0.114 | 0.096 | 0.095 | 0.094 | 0.118 |

## NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately describe the proposed Block-Union KV Selection and zero-copy paged execution design. The claimed accuracy preservation and speedup under chunked prefill are supported by the experiments and ablations in Section 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations of this work in Section 5.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: This paper does not present formal theoretical results, theorems, or proofs. The method is evaluated empirically.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide the method details, baseline settings, sparsity hyperparameters, hardware configurations, and benchmark settings needed to reproduce the main experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Code for reproducing the latency benchmark and RULER accuracy results is available at https://github.com/jiwonsong-dev/CompactAttention.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: We specify the evaluated models, benchmarks, baselines, sparsity hyperparameters, chunk sizes, batch sizes, tensor-parallel settings, and hardware configurations in Section 4.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: Appendix C reports the mean and standard deviation over five runs for the LLaMA-3.1-8B-Instruct latency measurements in Tables 7 and 8, and Qwen3-30B-A3B-Instruct-2507 latencies are averaged over three runs. The standard deviation captures run-to-run timing variation with fixed hardware and inputs. Accuracy results use fixed benchmark samples.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We report the GPU types, memory, tensor-parallel settings, batch sizes, and chunk sizes used for latency and accuracy experiments in Section 4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This research conforms to the NeurIPS Code of Ethics. It uses publicly available models and benchmarks and does not involve human subjects or sensitive personal data.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

Justification: We do not include a separate broader impact section because the work targets general-purpose inference efficiency rather than a specific application. Such efficiency improvements may reduce serving cost and energy use, but could also lower the cost of harmful LLM deployments.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: This work does not release new models, datasets, or other high-risk assets requiring safeguards. It uses publicly available models and benchmarks for evaluation.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We cite the original sources for the models, benchmarks, software frameworks, and baseline methods used in the paper.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: This work does not introduce or release new assets such as datasets or model checkpoints.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

Justification: This work does not involve research with human subjects, so IRB approval or equivalent review is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs were not used as an important, original, or non-standard component of the core method or experiments.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.