TAILORATTENTION: ACCELERATING LLM ATTENTION WITH LATENCY-AWARE CHUNK SCHEDULING

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern LLM services handle heterogeneous workloads with sequence lengths varying across tasks and requests. Chunked attention distributes KV-cache chunks across cooperative thread arrays (CTAs), making partitioning and scheduling critical to GPU utilization. Existing kernels, however, pipeline mainly within chunks, incurring boundary underutilization and phase-dependent CTA interference. Consequently, equal-sized chunks can have different latencies, making length-based scheduling unreliable and causing load imbalance and tail underutilization.

We present TailorAttention, a kernel–scheduler co-design for chunked attention. Its Hopper- and Blackwell-optimized kernels enable inter-chunk pipelining and reduce output-layout conversion through transposed tensor-core dataflows, yielding predictable chunk latencies captured by affine models. Guided by these estimates, a millisecond-scale scheduler jointly selects potentially unequal chunk sizes and CTA assignments while accounting for reduction overhead, with a provable additive bound. Across GPUs, models, and workloads, TailorAttention improves attention-kernel performance by up to 4.96× and reduces end-to-end latency by 27.1% over FlashAttention-3/4 and FlashInfer.

1 INTRODUCTION

Modern LLM services increasingly handle heterogeneous workloads—including question answering, long-document analysis, and agentic applications. Sequence lengths vary widely across tasks and among requests within the same task. When these requests run concurrently, attention kernels must handle this diversity. Since modern kernels partition KV caches into chunks distributed across cooperative thread arrays (CTAs), chunk partitioning and scheduling directly affect GPU utilization.

Existing attention kernels, however, do not handle this diversity well. FlashAttention-3/4 (FA-3/4) and FlashInfer (FI) primarily optimize pipelining within each chunk, leaving GPU resources underutilized at chunk boundaries. Moreover, boundary and steady-state phases use compute and memory resources differently, so interference among CTAs depends on which phases overlap. Since co-running chunks may spend different fractions of their execution in each phase, the interference, and thus chunk latency, becomes difficult to predict. Thus, equal-sized chunks can have very different latencies, making chunk length an unreliable latency proxy. Without accurate latency estimates, existing schedulers cannot reliably balance CTA completion times under heterogeneous workloads.

To address these limitations, we present TailorAttention (TA), a kernel–scheduler co-design for chunked attention. TA redesigns the tensor-core dataflow to reduce output-layout conversion overhead and enable efficient inter-chunk pipelining, thereby improving throughput and making chunk latency predictable. This predictability enables accurate estimates of chunk-processing and reduction latencies. Using these estimates, the scheduler jointly selects chunk sizes and CTA assignments, improving load balance and substantially reducing SM underutilization at the execution tail.

We make three main contributions. First, we show that inter-chunk overhead and phase-dependent interference make chunk latency unpredictable. Second, we design Hopper- and Blackwell-optimized kernels that improve inter-chunk pipelining and enable accurate latency modeling. Third, we formulate chunk partitioning and CTA assignment as a joint optimization problem and develop a fast scheduling algorithm with a provable bound. Across GPUs, models, and workloads, TailorAttention improves attention-kernel performance by up to 4.96× and reduces end-to-end latency by 27.1% over FlashAttention-3/4 and FlashInfer.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

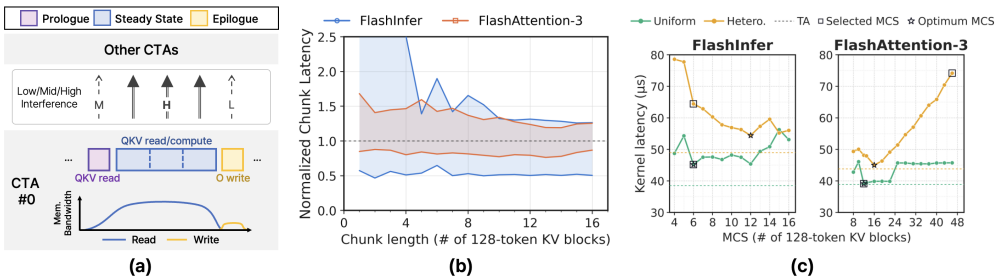


Figure 1: Difficulty of balancing CTA workloads in FA and FI. (a) Phase-dependent memory-bandwidth usage and interference during chunk processing. (b) Chunk lengths and their measured latencies. (c) FA/FI-selected versus optimal chunk sizes under uniform and heterogeneous workloads.

2 Background, Motivation, and Related Work

Attention with short queries—common in decoding and short prefill—is often memory-bound (1; 2; 3), motivating kernels that maximize HBM bandwidth utilization. FlashAttention (FA) tiles Q, K, and V to reduce memory traffic and improve parallelism (4; 5). Flash-Decoding and its successor partition a long KV cache into chunks, process them across CTAs, and reduce their partial outputs (6; 7). FlashInfer (FI) uses persistent CTAs and pipelines KV-tile loads with computation in each chunk (8).

Since the KV cache is processed in chunks, choosing chunk sizes and scheduling the resulting chunks across CTAs are critical for load balance. FI selects a chunk size that evenly partitions each sequence's KV cache across CTAs (8), whereas Stream-K and LeanAttention aggregate the KV lengths across the batch and evenly partition the total work (9; 10). For tree attention, DeFT flattens irregular KV trees and forms groups with similar token counts to balance work across thread blocks (11).

Beyond scheduling, FA-3 and FA-4 optimize intra-chunk pipelining for Hopper and Blackwell, respectively (12; 13). FA-3 uses producer-consumer warp specialization: producer warps load K and V tiles asynchronously through TMA, while consumer warpgroups execute WGMMA and softmax, overlapping data movement with computation (14). FA-4 adapts this pipeline to Blackwell using `tcgen05.mma` and TMEM accumulators (15), reducing register pressure and enabling additional overlap among tensor-core computation, softmax, and the output rescaling required by online softmax.

Workload Heterogeneity and Chunk Scheduling. Efficient chunked attention requires splitting sequences and scheduling the resulting chunks across CTAs, which becomes challenging when sequence lengths are diverse. Such diversity is increasingly common in modern LLM serving, where a single system may concurrently execute short-form question answering, chat, long-document summarization, and agentic workflows with substantially different context-length distributions(16; 17). Sequence lengths also vary within each workload because requests differ in prompt length, generated-response length, and reasoning effort(18), while multi-turn interactions cause context lengths to grow across turns(17; 19). This intra- and inter-task diversity creates three problems for chunked attention.

1. **Inter-chunk pipeline overhead.** FA and FI pipeline data movement and computation within chunks but provide limited overlap across chunk boundaries (Section 3). With diverse sequence lengths, balancing work across CTAs requires finer-grained and uneven chunking, producing more short chunks for which this boundary overhead accounts for a larger fraction of execution.
2. **Unpredictable chunk latency.** A chunk's prologue, steady state, and epilogue place different demands on GPU resources, causing interference among concurrent CTAs to vary across phases (Figure 1(a)). Consequently, equal-sized chunks can have widely different latencies (Figure 1(b)), making the chunk-length proxy used by FA and FI unreliable. Evenly partitioning KV lengths therefore fails to balance CTA completion times, causing underutilization.
3. **Inflexible chunk sizing.** Smaller chunks improve load balance but increase per-chunk overhead, while larger chunks reduce overhead but worsen imbalance. FA and FI use a single maximum chunk size (MCS), which works for similar sequence lengths but not diverse ones. As Figure 1(c) shows, the selected MCS can be far from optimal under heterogeneous workloads. Their H100 HBM bandwidth utilization consequently falls to 60.6% and 65.7%, respectively.

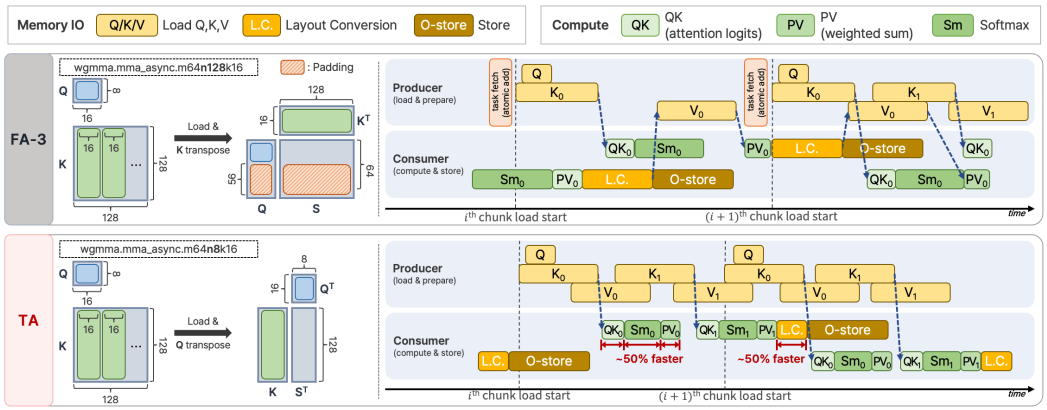


Figure 2: Comparison of the FA-3 (top) and TailorAttention (bottom) kernels. Left: WGMMA mappings for QK^T; PV is computed in the same manner. Right: Chunked-attention timelines illustrating TailorAttention's inter-chunk pipelining and faster QK^T and PV computations.

TailorAttention addresses the first two problems by redesigning the kernel for more efficient inter-chunk pipelining, reducing boundary overhead and enabling accurate chunk-latency modeling (Section 3). Using these estimates, its latency-aware scheduling algorithm jointly optimizes chunk partitioning and CTA assignment, addressing the third problem (Section 4). Together, these optimizations are effective at handling the intra- and inter-task length diversity of modern LLM serving.

3 Chunk Pipelining and Latency Modeling

TailorAttention restructures the attention kernel to improve throughput while making the execution latency of individual chunks more predictable. This section presents our chunk-pipelining design and introduces a model for estimating chunk latency. Section 4 then uses these estimates to jointly select chunk sizes and assign cooperative thread arrays (CTAs).

3.1 Efficient and Predictable Chunk Pipelining

As described previously, FA and FI overlap data movement with attention computation. Each CTA stages Q and streams K and V tiles from global memory into shared memory while computing QK^T, online softmax, and PV on previously loaded tiles. This pipeline operates efficiently within a chunk but provides limited overlap across chunk boundaries. Figure 2 (right) illustrates this limitation using a representative FA-3 execution timeline. Although their internal designs differ, FA-3, FA-4, and FI share this predominantly intra-chunk pipeline organization.

Extending the pipeline across chunks requires staging the next chunk's inputs before the current chunk completes. In FA-3, however, the output epilogue creates a long dependency at this boundary. FA-3 computes QK^T and PV using WGMMA, whose output remains in registers in an accumulator layout different from that of the output tensor. FA-3 must therefore rearrange and stage O in shared memory before TMA writes it to HBM; in our measurements, this path alone takes roughly one mainloop pipeline stage. Because FA-3 reuses this shared-memory region for both O and V, the next chunk cannot stage its V tiles until the current output is written back. Consequently, the epilogue delays the next chunk's pipeline.

FI and FA-4 use different scheduling and pipeline organizations: FI materializes chunk assignments in advance and dispatches chunks as separate CTA work units, whereas FA-4 reorganizes its pipeline around Blackwell-specific resources. Nevertheless, both primarily optimize execution within the current work unit and do not sufficiently overlap its output processing with input staging for the next chunk. Thus, although their precise dependencies differ from FA-3's shared V/O buffer, each chunk boundary exposes both a substantial portion of the current chunk's epilogue and the prologue overhead of the next chunk.

Beyond its direct overhead, limited inter-chunk overlap makes CTA interference, and thus chunk latency, highly variable. The prologue, steady state, and epilogue place different demands on L2 and HBM bandwidth, so contention depends on the phases and lengths of co-running chunks. Moreover, a delay in one chunk shifts the start of the next, changing which phases of other CTAs it overlaps with and the contention it experiences. Hence, equal-sized chunks can have very different latencies.

This phase-dependent interference is especially pronounced in mixed workloads, where a batch may combine tasks with different sequence-length distributions. When long and short sequences execute together, their chunks spend different fractions of time in boundary and steady-state phases, continually changing the contention experienced by co-running CTAs. Chunk latency therefore becomes difficult to predict from chunk length alone.

TailorAttention Kernel Design. Our kernel enables efficient inter-chunk pipelining and substantially reduces the output-layout conversion overhead in each chunk's epilogue. That is, TailorAttention allocates separate shared-memory buffers for V and O, allowing the next chunk's V tiles to be staged while the current chunk's O is rearranged and written back. To offset the additional shared-memory footprint and further reduce this conversion overhead, we redesign short-query attention around a transposed computation. This optimization is particularly important for increasingly prevalent multi-turn workloads, in which decoding and short prefill recur across turns.

More specifically, Hopper's FP16/BF16 WGMMA instructions use $m64nNk16$ shapes(20), with a fixed $M = 64$ and a configurable N in multiples of eight. Directly computing $S = QK^\mathsf{T}$ maps the query length q to M, wasting most of the tile when q is short as shown in Figure 2 (left). The same inefficiency appears in PV, whose padded WGMMA output contains $64 \times d$ elements (where d is the dimension of a head) even for a few queries, increasing both its accumulator footprint and the amount of data rearranged before writeback.

TA instead computes $S^\mathsf{T}{=}KQ^\mathsf{T}$, applies softmax along the KV dimension to obtain P^T, and computes $O^\mathsf{T}{=}V^\mathsf{T}P^\mathsf{T}$. This maps the short query dimension to N, reducing the padded PV output to $d \times \{8, 16\}$ elements for $q \leq 16$. No explicit transpose is needed: the column-major layout of O^T matches the row-major layout of O; only the smaller WGMMA fragment must be rearranged and staged for writeback. The smaller O buffer enables separate V/O storage and reduces the inter-chunk bubble.

For Blackwell, we apply a similar optimization to our FA-4-based implementation, which instead uses `tcgen05.mma`. The transposed formulation maps the short query dimension to N, reducing padded computation within the MMA tiles and improving tensor-core utilization. On both architectures, these optimizations smooth compute and memory demand across pipeline stages, making chunk latency more predictable and enabling the latency-estimation model described in Section 3.2.

3.2 Latency Estimation Model

Chunked attention kernels typically follow a two-stage execution structure. The chunk-processing stage independently computes a partial attention output and the log-sum-exp (LSE) normalization term for each chunk. The reduction stage then combines the partial results from all chunks for the same sequence to produce the final attention output. Because the two stages have different computational characteristics, we use separate affine models to estimate their execution latencies. We present the estimation model mainly in the context of decoding and briefly discuss the prefill case.

For the chunk-processing stage, we model the execution latency of a chunk as $T(L) = aL + b$, where L denotes the chunk length measured in 128-token blocks. This affine model separates the steady-state processing cost, which scales with the chunk length, from the fixed overhead incurred by each chunk. That is, the slope a represents the marginal latency of processing an additional 128-token block and reflects the effective throughput of the pipelined kernel. The intercept b captures length-independent costs, such as the prologue and epilogue overheads of the processing pipeline.

For the reduction stage, we model the latency for a sequence as $R(K) = cK + d$ for $K \geq 1$ and set $R(0) = 0$, where K is the number of additional chunks to be merged, i.e., one less than the total number of chunks for that sequence. For prefill, the workload is also partitioned along the query dimension, independently of KV-chunk reduction, so the same reduction model applies to each query group. Because the chunk-processing stage has already transformed each chunk into a partial attention output and its corresponding LSE term, the original chunk lengths do not affect the

Table 1: Notation for the Mixed-Integer Linear Program.

Symbol	Description	Symbol	Description
$T,\ R$	chunk-processing, reduction makespan	$q_{ij} \in \mathbb{Z}_{\geq 0}$	chunk length of seq. i on CTA j
$N,\ M$	number of sequences and CTAs	$y_{ij} \in \{0,1\}$	1 if seq. i has a chunk on CTA j
L_i	length of seq. i (128-token blocks)	$k_i = \sum_j y_{ij}$	number of chunks of seq. i
c_{KV}	fixed per-chunk overhead	$s_i \in \{0,1\}$	1 if seq. i is split ($k_i \geq 2$)
$r_0,\ r_1$	fixed and per-merge reduction cost (d and c in $cK + d$)	$z_{ij} \in \{0,1\}$	1 if reduction of i runs on CTA j
		$w_{ij} \geq 0$	linearization of $z_{ij}(k_i - 1)$

reduction latency. The slope c represents the marginal latency of incorporating one additional chunk, whereas the intercept d captures the fixed overhead of the reduction stage.

We fit the latency-model coefficients via linear regression on profiled measurements. Across decoding, speculative decoding, and prefill with short queries, the fitted models achieve an R^2 of 0.997 with the coefficient of variation below 5.74%. Applying the same method to FI and FA yields lower R^2 values (FI:0.717 and FA:0.929) and 2–2.4$\times$ larger latency variability. For prefill with longer queries ($q > 128$), accurate latency estimation is less critical because tail underutilization accounts for only a small fraction of execution time; we therefore do not target this regime. The coefficients are fitted only once per GPU model and packaged with the deployed kernels, allowing the chunk scheduler in Section 4 to use them without online profiling.

4 Latency-Aware Chunk Scheduling

Using the latency estimates from Section 3, our scheduler statically partitions each sequence into potentially unequal chunks and assigns them to CTAs. Finer-grained chunking improves load balance and reduces tail underutilization, but incurs additional pipeline and reduction overheads. The scheduler therefore jointly optimizes the partitioning and CTA assignment to minimize the attention makespan. We formulate this optimization as a mixed-integer linear program and describe an efficient solution method in the following subsections.

4.1 Mixed-Integer Linear Programming Formulation

Let a batch consist of N sequences processed by M persistent CTAs, which act as workers. We express a sequence length in 128-token blocks and allow each sequence to be partitioned into potentially unequal chunks assigned across the CTAs. Table 1 describes the notation in our formulation.

Using this notation, we formulate the joint chunk-partitioning and scheduling problem as the following mixed-integer linear program (MILP). The objective is to minimize the total attention latency $J = T + R$, comprising the chunk-processing makespan T and the reduction-stage latency R:

$$
\begin{aligned}
\text{minimize} \quad & J = T + R \\
\text{s.t.} \quad & \textstyle\sum_i \big(q_{ij} + c_{\mathrm{KV}}\, y_{ij}\big) \leq T, \qquad \sum_i \big(r_0\, z_{ij} + r_1\, w_{ij}\big) \leq R && \forall j, \\
& \textstyle\sum_j q_{ij} = L_i, \qquad y_{ij} \leq q_{ij} \leq L_i\, y_{ij} && \forall i, j, \\
& s_i \leq k_i - 1 \leq (M-1)\, s_i, \qquad \textstyle\sum_j z_{ij} = s_i && \forall i, \\
& w_{ij} \geq k_i - 1 - (M-1)(1 - z_{ij}) && \forall i, j.
\end{aligned}
$$

All latencies are normalized by the per-block chunk-processing latency. Solving this MILP exactly can be too costly for online scheduling. In the next subsection, we present an efficient algorithm that finds a schedule whose objective value is at most a fixed amount above the optimum. The algorithm schedules hundreds of sequences across hundreds of CTAs within a few milliseconds and can run concurrently with the current-step forward pass.

4.2 A Fast Scheduling Algorithm with an Additive Optimality Bound

Our scheduling algorithm defines the search space for the chunk-processing makespan using lower and upper bounds. Starting from the lower bound, it examines candidate makespans in increasing order and attempts to construct a feasible chunk partition and CTA assignment for each candidate.

To obtain the upper bound, we construct a feasible schedule without splitting any sequence. That is, we consider the sequences in non-increasing order of length and assign each one to the CTA with the smallest current workload. Let T_u denote the resulting chunk-processing makespan. Because no reduction is required, the total latency is $J_u = T_u$.

To derive the lower bound, we observe that every sequence must be processed as at least one chunk. Thus, any feasible schedule incurs at least $\sum_i L_i + N c_{\mathrm{KV}}$ units of chunk-processing work, including one chunk overhead for each sequence. Even if this workload could be perfectly balanced across the M CTAs, the makespan would be $T_\ell = \frac{\sum_i L_i + N\, c_{\mathrm{KV}}}{M}$. Although such perfect balancing may be infeasible, T_ℓ is a valid lower bound on the optimal chunk-processing makespan.

Starting with $T_\tau = \lceil T_\ell \rceil$, the algorithm applies a fast constructive test to each candidate makespan T_τ, although failure does not rule out a different feasible schedule with the same makespan. For each candidate, we sort the sequences by length in descending order, initialize the remaining capacity of every CTA to T_τ, and visit the CTAs in index order. We then construct a schedule as follows:

1. For the current sequence i and CTA j, create a new chunk and deduct its fixed overhead c_{KV} from the CTA's remaining capacity. If the CTA cannot accommodate a nonempty chunk, advance to the next CTA.
2. Assign as much of the unassigned portion of sequence i as fits. If the entire portion fits, advance to the next sequence; if not, fill the current CTA and continue the remaining portion on CTA $j + 1$.
3. Repeat until all sequences are assigned or all CTAs are exhausted. In the former case, record the constructed schedule; in the latter, discard the failed construction. Then proceed to the next candidate makespan.
4. After evaluating all candidate makespans, return the schedule with the smallest $J = T + R$.

To compute R, we create one reduction task for each split sequence i, with latency $r_0 + r_1(k_i - 1)$. Using Longest Processing Time first (LPT), we assign these tasks in descending order of latency to the least-loaded reduction CTA. The maximum resulting CTA workload gives R.

Optimality Bound. While we select a schedule based on its total latency $J = T + R$, candidate schedules are constructed by searching over the chunk-processing makespan T. We show that this search produces a feasible candidate whose T is within a constant additive gap of the optimum.

Theorem 1. *Let $T^\star$ denote the optimal chunk-processing makespan. The candidate set generated by our algorithm contains a feasible schedule with makespan $\widehat{T}$ satisfying*

$$\widehat{T} - T^\star \le c_{\mathrm{KV}} + 1.$$

Thus, for a fixed kernel configuration, the gap is independent of the number and lengths of sequences.

Proof sketch. Recall that $T_\ell = (\sum_i L_i + N c_{\mathrm{KV}})/M$ is a lower bound on $T^\star$, while T_u is the makespan of a feasible no-split schedule. If $T_u \le \lceil T_\ell \rceil + c_{\mathrm{KV}}$, this schedule already satisfies the theorem. Otherwise, the search considers $T = \lceil T_\ell \rceil + c_{\mathrm{KV}}$.

We show that $T = \lceil T_\ell \rceil + c_{\mathrm{KV}}$ admits a feasible schedule using a wrap-around construction. We concatenate the baseline workloads of all sequences and place them sequentially across the CTAs, assigning each CTA a main capacity of $\lceil T_\ell \rceil$. Since $M\lceil T_\ell \rceil \ge M T_\ell = \sum_i L_i + N c_{\mathrm{KV}}$, the main regions can accommodate the entire baseline workload. When a CTA boundary splits a sequence, its remainder becomes a new chunk on the next CTA and incurs an additional c_{KV} overhead. Because each boundary splits at most one sequence, reserving c_{KV} additional capacity per CTA is sufficient. Thus, the algorithm constructs a schedule satisfying

$$\widehat{T} \le \lceil T_\ell \rceil + c_{\mathrm{KV}}.$$

Since $T_\ell \le T^\star$, we obtain

$$\widehat{T} - T^\star \le \lceil T_\ell \rceil - T_\ell + c_{\mathrm{KV}} \le c_{\mathrm{KV}} + 1. \qquad \square$$

In practice, we search over candidate values of T, derive the resulting reduction latency R, and select the candidate minimizing $J = T + R$. This one-dimensional search accounts for reduction overhead while remaining fast enough for online scheduling.

5 Evaluation

Setup. We conduct all experiments mainly on two machines, one with four NVIDIA H100 NVL GPUs each with 94GB of HBM and Intel Xeon 6515P, and another with one NVIDIA B300 SXM6 each with 288GB of HBM and Intel Xeon 6767P. The software stack consists of NVIDIA driver 580.159.03, CUDA runtime 13.0, CUDA Toolkit/`nvcc` 13.3, cuDNN 9.20.0, and PyTorch 2.13.0+cu130.

We evaluate Llama 3 (21), Qwen3 (22) and Gemma 3 (23) model families, as well as `gpt-oss-120b`, using BF16 for attention and model weights. We derive request sequences from four datasets that span a wide range of prompt and output lengths: SQuAD (24) for question answering, GovReport (25) in LongBench (26) for long-document summarization, ShareGPT (27) for chat, and Codex agent traces from SWE-Bench Pro (28; 29) for agentic workloads.

We compare TailorAttention against FA-3 from FlashAttention 3.0.0, FA-4 from FlashAttention 4.0.0b18 and FlashInfer 0.6.18 under identical model configurations and inputs. We perform 5 warm-up iterations followed by 10 measured iterations and report the mean and variability.

5.1 Attention Kernel Microbenchmarks

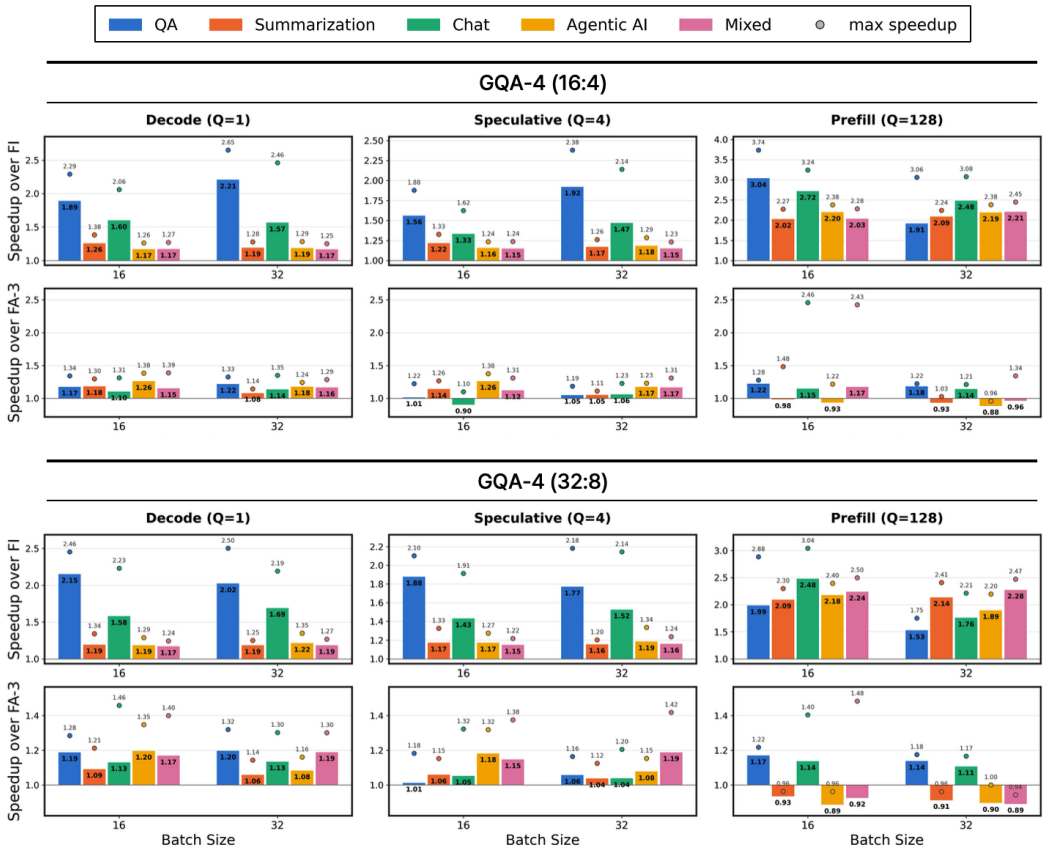


Figure 3: Attention-kernel performance on H100 across four dataset-derived workloads and their heterogeneous mix for decoding ($q = 1$), speculative decoding ($q = 4$), and short prefill ($q = 128$). Bars show FA-3 and FI latencies normalized to TA (TA $= 1$), for each GQA configuration; higher is better.

We microbenchmark attention kernels on the four dataset-derived workloads and a heterogeneous mix. Each run maintains 16 active requests, replacing completed requests with samples using their dataset-provided input and output lengths. For the heterogeneous workload, the source dataset rotates among all four in round-robin order. We measure kernel latency for decoding, speculative decoding with a draft length of four, and short prefill with 128 query tokens.

We compare TA against FA-3 and FI on H100, and FA-4 and FI on B300. We evaluate $(H_q, H_{\mathrm{KV}}) \in \{(4,1),(16,4),(16,8),(32,8)\}$, covering GQA configurations across Llama and Qwen models and tensor-parallel degrees. All kernels use head dimension of 128, identical BF16 inputs, and the same KV-cache lengths. We report average latency over 10 runs.

Figure 3 reports the H100 results; B300 results follow similar trends and are omitted for brevity. We normalize FA-3 and FI latencies to TA latency ($\mathrm{TA} = 1$), so each bar directly represents TA's speedup over the corresponding baseline.

Across all workloads, TA achieves up to $3.74\times$ and $2.46\times$ speedups over FA-3 and FI, respectively, and outperforms both in 53 of 60 configurations. The gains are particularly large for question answering and the heterogeneous workload. Question answering primarily benefits from TA's inter-chunk pipelining, whereas the heterogeneous workload benefits from its load-balanced scheduling; Section 5.2 provides a detailed breakdown.

5.2 Performance Breakdown

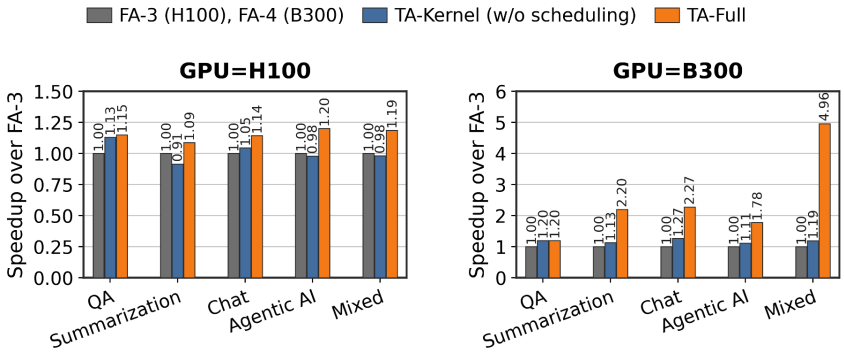


Figure 4: Performance gains from our kernel optimization (TA-Kernel) and its combination with latency-aware scheduling (TA-Full)

We separately measure the performance gains from TA's attention-kernel optimizations and scheduling algorithm. We compare three configurations: the architecture-specific baseline (FA-3 on H100 and FA-4 on B300), *TA-Kernel*, which enables only our kernel optimizations, and *TA-Full*, which additionally enables latency-aware scheduling. We use the same four dataset-derived workloads and heterogeneous mix as in Section 5.1.

Figure 4 presents the breakdown on H100 (left) and B300 (right). On H100, TA-Kernel achieves up to 13% speedup over FA-3, with the largest gains on question answering due to improved inter-chunk pipelining. TA-Full provides a further 20% speedup, benefiting the heterogeneous workload most because its diverse sequence lengths create greater scheduling imbalance.

B300 shows a similar pattern: the Blackwell kernel optimization achieves up to 27% speedup over FA-4, while latency-aware scheduling adds up to $4.96\times$. Overall, the kernel improves per-chunk execution efficiency, while the scheduler converts predictable chunk latency into better load balance.

5.3 Scheduling Efficiency

We evaluate TA's scheduling efficiency against FA-3 and FI using average SM idle time. Conventional load balance efficiency (LBE) measures the fraction of available execution time spent on useful work. However, chunk-boundary and reduction overheads may be counted as busy time, inflating LBE despite increasing latency. We therefore report SM idle time directly.

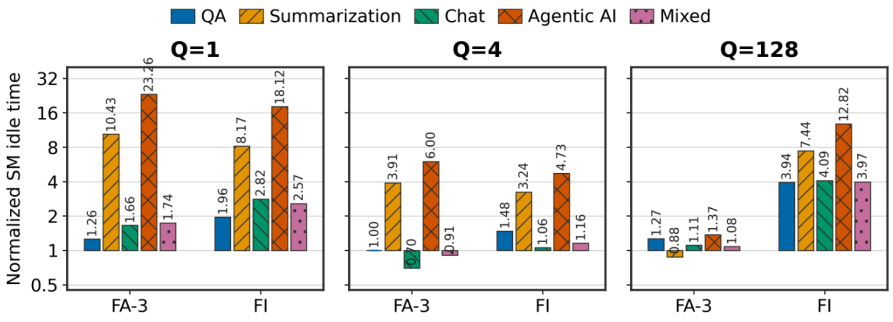


Figure 5: Scheduling inefficiency of FA-3 and FI, measured as mean SM idle time normalized to TA.

Figure 5 reports the average SM idle times of FA-3 and FI normalized to TA (TA $= 1$) across the four dataset-derived workloads and heterogeneous mix for $q \in \{1, 4, 128\}$. TA has lower idle time in 27 of 30 comparisons, reducing it by up to $20\times$. In the three cases where FA-3 idles less, such as chat with $q = 4$, its reduction stage is 1.76–1.84$\times$ longer than TA's, and TA still achieves lower overall attention latency. The scheduler incurs only a few hundred microseconds of overhead (Table 2), which can be hidden by the previous step's forward pass or the first few layers of the current step.

Table 2: Scheduling overhead (μs).

Dataset	Mean	P90	P99	Dataset	Mean	P90	P99
QA	357.22	432.66	882.30	Agentic AI	511.67	615.75	869.17
Summarization	484.95	519.80	604.44	Mixed Workload	404.28	559.50	757.56
Chat	388.63	487.88	508.02	**Overall**	429.35	560.41	775.64

5.4 End-to-End Serving Performance

We evaluate end-to-end serving performance using the ConsistentChat (30) multi-turn workload and the gpt-oss-120b model. We integrate TA into SGLang and run it on a single NVIDIA H100 GPU with continuous batching and a batch size of 16. We compare full TA against FA-3 under identical configurations. We choose FA-3 because it is SGLang's default backend on H100 and outperforms FI for query lengths of 128 or more in our kernel microbenchmarks.

FA-3 achieves 1,965.0 tokens/s with an average per-turn latency of 1,465.1 ms. TA increases throughput to 2,694.8 tokens/s, which is a 37.1% improvement, while reducing average per-turn latency to 1,067.6 ms, a 27.1% reduction. These results demonstrate that TA's kernel and scheduling optimizations translate into substantial end-to-end gains for continuous multi-turn serving.

6 Conclusion

We presented TailorAttention, a kernel–scheduler co-design for chunked attention under heterogeneous LLM workloads. Existing kernels pipeline within chunks, exposing boundary overhead and phase-dependent interference that make chunk latency unpredictable. TailorAttention's Hopper- and Blackwell-optimized tensor-core dataflows enable inter-chunk pipelining and reduce output-layout conversion. The resulting predictable latencies are accurately captured by simple affine models.

TailorAttention jointly selects chunk sizes and CTA assignments while accounting for reduction overhead. Its scheduling algorithm provides a provable additive bound and substantially reduces tail underutilization. Across GPUs, models, and workloads, TailorAttention improves attention-kernel performance by up to 4.96$\times$ over FlashAttention-3/4 and FlashInfer. In end-to-end multi-turn serving, it reduces average turn latency by 27.1% over SGLang's default FlashAttention-3 backend.

REFERENCES

[1] Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

[2] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 5, pages 606–624, 2023.

[3] Pol G. Recasens, Ferran Agullo, Yue Zhu, Chen Wang, Eun Kyung Lee, Olivier Tardieu, Jordi Torres, and Josep Ll. Berral. Mind the memory gap: Unveiling GPU bottlenecks in large-batch LLM inference. In *2025 IEEE 18th International Conference on Cloud Computing (CLOUD)*, pages 277–287. IEEE, 2025.

[4] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pages 16344–16359, 2022.

[5] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, 2024.

[6] Tri Dao, Daniel Haziza, Francisco Massa, and Grigory Sizov. Flash-Decoding for long-context inference. PyTorch Blog. `https://pytorch.org/blog/flash-decoding/`, October 2023. Accessed: 2026-09-26.

[7] Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Yuhan Dong, and Yu Wang. FlashDecoding++: Faster large language model inference with asynchronization, flat GEMM optimization, and heuristics. In *Proceedings of Machine Learning and Systems*, volume 6, pages 148–161, 2024.

[8] Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. FlashInfer: Efficient and customizable attention engine for LLM inference serving. In *Proceedings of Machine Learning and Systems*, volume 7, 2025.

[9] Muhammad Osama, Duane Merrill, Cris Cecka, Michael Garland, and John D. Owens. Stream-K: Work-centric parallel decomposition for dense matrix-matrix multiplication on the GPU. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, PPoPP '23, pages 429–431. Association for Computing Machinery, 2023.

[10] Rya Sanovar, Srikant Bharadwaj, Renée St. Amant, Victor Rühle, and Saravan Rajmohan. LeanAttention: Hardware-aware scalable attention mechanism for the decode-phase of transformers. In *Proceedings of Machine Learning and Systems*, volume 7, 2025.

[11] Jinwei Yao, Kaiqi Chen, Kexun Zhang, Jiaxuan You, Binhang Yuan, Zeke Wang, and Tao Lin. DeFT: Decoding with flash tree-attention for efficient tree-structured LLM inference. In *International Conference on Learning Representations*, 2025.

[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. In *Advances in Neural Information Processing Systems*, volume 37, pages 68658–68685, 2024.

[13] Ted Zadouri, Markus Hoehnerbach, Jay Shah, Vijay Thakkar, and Tri Dao. FlashAttention-4: Algorithm and kernel pipelining co-design for asymmetric hardware scaling. In *Proceedings of Machine Learning and Systems*, volume 8, 2026.

[14] NVIDIA. NVIDIA H100 tensor core GPU architecture. Whitepaper, NVIDIA Corporation, 2022.

[15] NVIDIA. NVIDIA Blackwell architecture technical brief. Technical brief, NVIDIA Corporation, 2024.

[16] Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation — a KVCache-centric architecture for serving LLM chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pages 155–170, Santa Clara, CA, 2025. USENIX Association.

[17] Kan Zhu, Mathew Jacob, Chenxi Ma, Yi Pan, Stephanie Wang, Arvind Krishnamurthy, and Baris Kasikci. TraceLab: Characterizing coding agent workloads for LLM serving. *arXiv preprint arXiv:2606.30560*, 2026.

[18] OpenAI. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

[19] Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. Cost-Efficient large language model serving for multi-turn conversations with CachedAttention. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 111–126, Santa Clara, CA, 2024. USENIX Association.

[20] NVIDIA. Parallel thread execution ISA version 9.4. https://docs.nvidia.com/cuda/archive/13.2.0/parallel-thread-execution/index.html#ptx-isa-version-9-2. Accessed: 2026-09-26.

[21] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, et al. The llama 3 herd of models, 2024.

[22] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. Qwen3 technical report, 2025.

[23] Gemma Team. Gemma 3 technical report, 2025.

[24] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics.

[25] Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1419–1436, Online, June 2021. Association for Computational Linguistics.

[26] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[27] ShareGPT_Vicuna_unfiltered. https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered, 2023. Accessed: 2026-09-26.

[28] Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-bench Pro: Can AI agents solve long-horizon software engineering tasks? In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, Seoul, South Korea, 2026. PMLR.

[29] Inferact. codex_swebenchpro_traces. https://huggingface.co/datasets/Inferact/codex_swebenchpro_traces, 2026. Accessed: 2026-09-26.

[30] Jiawei Chen, Xinyan Guan, Qianhao Yuan, Mo Guozhao, Weixiang Zhou, Yaojie Lu, Hongyu Lin, Ben He, Le Sun, and Xianpei Han. Consistentchat: Building skeleton-guided consistent multi-turn dialogues for large language models from scratch. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 8426–8452, 2025.

A Disclosure of AI Assistance

In this work, we used generative AI tools to formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in writing proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, assist with translation, and clean and reformat datasets. We did not use generative AI tools to generate synthetic datasets, help develop theoretical models or conceptual frameworks, support qualitative or thematic data analysis, or interpret results. Formulating questions for surveys or interviews is not applicable to this work. Additionally, we used generative AI tools to create or modify scientific figures or images, create or edit software code, create artifacts, brainstorm, source and search for information, edit the paper to improve readability, identify relevant literature, and format references. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including any text, claims, or artifacts produced with the aid of generative AI.