Accelerating Long-Context LLM Prefill via Layer-wise Progressive Token Pruning in Local Deployment

Zhongxiang Wei[1], **Zhaohan Wang**[1], **Zhixiong Zhang**[1], **Jin Zhao**[1], **Xiaoming Fu**[2,3*]
[1]College of Computer Science and Artificial Intelligence, Fudan University, Shanghai, China
[2]Institute of Computer Science, University of Göttingen, Göttingen, Germany
[3]Shanghai Key Laboratory for Intelligent Information Processing, Fudan University, Shanghai, China

Abstract

Long-context inference is increasingly important for privacy-sensitive local LLM deployment. In this setting, the prefill stage dominates Time-to-First-Token (TTFT), yet existing acceleration methods struggle to balance speed and quality. Sparse-attention methods accelerate only the attention module, leaving Feed-Forward Network (FFN) costs unchanged, while auxiliary-model-based compression often compromises quality under tight memory budgets. Recent layer-wise token pruning methods reduce both attention and FFN computation, but they typically rely on heuristic pruning schedules or token-recovery mechanisms that introduce decoding overhead. We propose FastPrefill, a recovery-free layer-wise token pruning system for efficient local long-prompt inference. FastPrefill employs an offline data-driven optimizer to determine layer-wise pruning schedules that maximize fidelity under a latency constraint, and executes these schedules via an algorithm–system co-design supporting head-specific sparse attention at runtime, where each head dynamically attends to its own selected key/value tokens. Evaluations on LongBench and RULER show that FastPrefill maintains comparable inference quality while achieving up to 2.13× TTFT speedup and 3.03× end-to-end speedup over state-of-the-art baselines.

1 Introduction

Large Language Models (LLMs) [29] are increasingly deployed on personal computers and edge devices for privacy-sensitive applications [22, 23], such as document analysis and agentic workflows [44]. In such local deployments, LLM inference typically runs on a single consumer-grade GPU with a batch size of 1, processing prompts sequentially [34]. This setting makes efficient long-prompt processing particularly challenging: as prompt length grows, the prefilling stage becomes increasingly expensive, substantially increasing Time-to-First-Token (TTFT). Since generating the first output token requires processing the entire prompt, TTFT is a key factor in perceived responsiveness. However, existing methods fail to adequately balance prefill acceleration and inference quality in resource-constrained local environments.

First, **sparse-attention methods** [18, 40, 43] may provide limited prefill acceleration in local settings. They optimize only the attention module, leaving the Feed-Forward Network (FFN) computation unreduced. In memory-bound regimes typical of consumer GPUs (e.g., RTX 4090), FFN latency is comparable to attention latency for prompts ranging from several thousand to tens of thousands of tokens. This motivates **token pruning**, which directly reduces the computational load of both attention and FFN components.

Second, **auxiliary-model-based context compression** [17, 26] often degrades inference accuracy. These methods rely on a small assistant model to identify and discard unimportant tokens before

*Corresponding authors: Jin Zhao (jzhao@fudan.edu.cn) and Xiaoming Fu (fu@cs.uni-goettingen.de).

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

full-model inference. Under tight local memory constraints, practitioners are forced to use smaller assistant models, which compromises compression quality. This limitation underscores the need to guide compression decisions using signals from the target model itself.

Third, recent **layer-wise token pruning works** [9, 28, 41] reduce prefill computation, but rely on hand-crafted pruning schedules (e.g., fixed pruning layers or manually specified pruning ratios), rather than optimizing for the latency-accuracy trade-off. For example, SlimInfer [28] retains a constant number of tokens at certain layers regardless of prompt length. While this reduces prefill computation, it relies on token recovery mechanisms to preserve accuracy, introducing additional CPU–GPU data movement and synchronization costs in the decoding phase.

To address these challenges, we propose FastPrefill, an efficient, recovery-free, and progressive token pruning method for single-GPU local deployment. FastPrefill consists of two key components.

(1) Offline Data-Driven Optimizer: We formulate the search for a layer-wise pruning schedule as an optimization problem. However, evaluating each candidate with full-model inference is prohibitively expensive. To address this, we build a lightweight surrogate objective from attention scores collected on a calibration dataset. We further construct independent per-layer optimization variables according to the progressive pruning process, making the search tractable via evolutionary optimization (CMA-ES) [13].

(2) Algorithm–System Co-Design: We translate the optimized pruning schedule into wall-clock speedup through a custom runtime pipeline. At runtime, FastPrefill uses lightweight attention scores to decide which tokens should be retained under the layer-wise schedule. This is non-trivial because directly materializing full attention scores is bandwidth-intensive and may cause out-of-memory (OOM) errors for long prompts. In addition, standard dense attention kernels cannot exploit head-specific dynamic sparsity, while naive sparse implementations introduce high indexing and synchronization overhead. To address these challenges, FastPrefill recomputes lightweight attention scores online, uses them to select head-specific key-value (KV) tokens, and stores the retained KV tokens in a compact layout. It then performs attention with a customized boundary-aware sparse kernel, enabling the pruning decisions to translate into practical prefill acceleration.

We evaluate FastPrefill against seven baselines on LongBench [3] and RULER [16]. Results show that FastPrefill achieves up to 2.13× TTFT speedup and 3.03× end-to-end speedup while maintaining comparable inference quality. **In summary, FastPrefill makes three contributions.** (1) Problem framing. Based on our motivations, we formulate recovery-free prefill acceleration as a latency-constrained optimization problem, rather than using hand-crafted pruning schedules. (2) Novel design. We couple schedule optimization with executable sparse prefill, so the searched schedules satisfy latency constraints and map to dynamic, head-specific, causal KV sparsity at runtime. (3) Extensive evaluations across diverse benchmarks validate our design and demonstrate that FastPrefill provides a superior efficiency-quality trade-off compared to state-of-the-art baselines.

2 Preliminaries and Motivation

2.1 Self-Attention and Prefill Cost

An LLM consists of stacked Transformer layers [36]. At a layer, the input hidden state X is projected into query, key, and value representations, denoted by Q, K, and V. The key and value vectors are stored in the KV cache for subsequent decoding. The layer then executes self-attention computation, i.e., $\text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$, followed by an FFN to produce the hidden states for the next layer.

Optimizing attention alone is insufficient for substantial prefill acceleration in local deployment. We profiled the latency of Self-attention (implemented with FlashAttention [7]) and the FFN within a Llama3.1-8B layer [11] on an RTX 4090. As shown in Fig. 1, FFN cost is comparable to attention cost over a wide range of sequence lengths (4K – 60K) in memory-bound settings.

Token pruning reduces the cost of both attention and FFN. Specifically, let $Q \in \mathbb{R}^{n \times d}$. If we retain only $\tilde{n}$ salient tokens ($\tilde{n} < n$), both attention and FFN are computed over fewer tokens, thereby reducing the workload of both components. Note that we do not directly reduce X. Instead, we prune the token dimension of Q, K, and V to better preserve inference quality. In the following sections, unless otherwise specified, a query token, a key token, and a value token refer to a row vector in Q, K, and V, respectively.

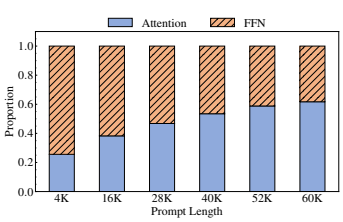

Figure 1: Latency contribution of Attention vs. FFN across different prompt lengths.

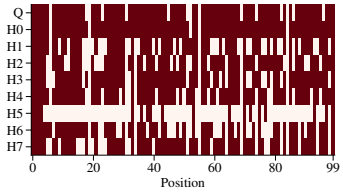

Figure 2: QKV token retention patterns in a layer (50% sparsity). Red indicates retained positions.

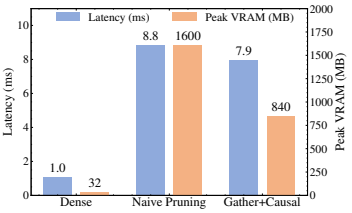

Figure 3: Latency and peak VRAM of different attention implementations.

2.2 Design Challenges

Since prefill cost is accumulated layer by layer through self-attention and FFN, progressively pruning tokens across layers provides a natural way to accelerate prefill. However, it introduces two key challenges:

Challenge 1: Determining the layer-wise pruning schedule under a desired efficiency-quality trade-off. Consider an L-layer LLM f_{LLM}. Given an input sequence q, let $\mathcal{R} = \{r_1, r_2, \ldots, r_L\}$ denote the layer-wise pruning schedule, where r_i is the ratio of cumulative pruned tokens after layer i to the prompt length. Thus $0 \leq r_1 \leq r_2 \leq \cdots \leq r_L < 1$. Given a latency ratio target α, our goal is to find a schedule $\mathcal{R}$ that preserves model quality while satisfying the latency constraint, i.e., $f_{\text{LLM}}(q \mid \mathcal{R}) \approx f_{\text{LLM}}(q)$, subject to $\mathcal{G}(\mathcal{R}) = \frac{T_{\text{prefill}}(\mathcal{R})}{T_{\text{prefill}}(\mathbf{0})} \approx \alpha$, where $\mathcal{G}$ is the latency function.

This is challenging because: 1) The schedule is globally coupled, i.e., pruning at layer i shortens the sequence processed by all subsequent layers. Thus, unlike static weight pruning, the latency reduction and quality drop caused by r_i cannot be evaluated independently. 2) The objective is expensive to evaluate directly. Searching for optimal schedules requires comparing the outputs of the pruned and full models across numerous candidates, which involves repeated and costly full-model inference. 3) Token pruning is fundamentally implemented via discrete token selection, rendering the search space non-differentiable and unsuitable for standard gradient-based optimization.

Challenge 2: Translating pruning into wall-clock speedup. Different attention heads exhibit distinct attention patterns [18, 40], thus enforcing a shared retained KV set limits pruning efficacy. Fig. 2 visualizes retained query tokens and per-head KV tokens in a Llama3.1-8B layer. The example is a 4K-token prompt from LongBench [3], with a pruning ratio of 50%. It shows that the sparsity pattern is dynamic, irregular, and head-specific, which creates two system challenges: 1) *Memory overhead:* Runtime token selection can introduce substantial memory overhead. Token importance is estimated from attention scores, but naively materializing full attention-score matrices for long prompts incurs quadratic memory cost and can easily trigger OOM errors. 2) *Kernel inefficiency:* the selected query tokens and per-head KV tokens no longer form a regular causal attention layout, making the resulting sparsity difficult to exploit with existing kernels. (1) Standard dense attention, such as FlashAttention [7], can process the original dense layout, but it cannot skip pruned token interactions and therefore provides little wall-clock speedup. (2) A naive masked implementation applies an $n \times n$ causal mask in eager attention [36], so pruned interactions are masked logically rather than removed physically. As shown in Fig. 3 (under the same experimental setup as Fig. 2), this substantially increases latency and peak VRAM usage due to the I/O and memory overhead of materializing large attention-score matrices. (3) A gathered-KQV implementation first compacts the retained query, key, and value vectors and then constructs a smaller causal mask over the compacted tensors. However, it still incurs substantial latency and memory overhead.

3 Methods

We present the design of FastPrefill in Fig. 4. To address Challenge 1, FastPrefill constructs a surrogate objective from a calibration dataset and optimizes the layer-wise pruning schedule to balance efficiency and quality (§3.1). To address Challenge 2, FastPrefill uses an online execution pipeline that recomputes lightweight attention scores, derives Q/KV retention indices, compacts retained KV vectors into a contiguous cache, and executes a boundary-aware sparse attention kernel to produce the output for the subsequent FFN layer (§3.2).

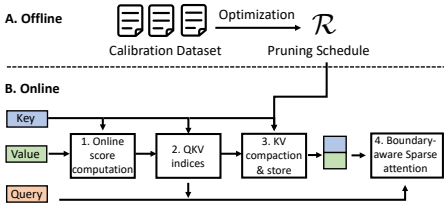

Figure 4: The workflow of offline optimization and online attention workflow in FastPrefill.

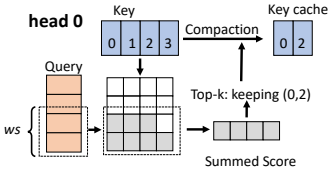

Figure 5: Key token selection and compaction.

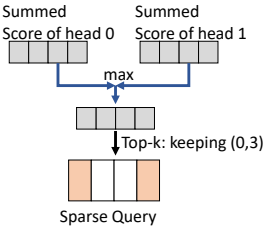

Figure 6: Query token selection.

3.1 Offline Pruning Schedule Optimization

3.1.1 Surrogate Objective Formulation

To make the problem in challenge 1 tractable, we relax it into a penalty-based surrogate objective in Eq. 1. It includes two terms: a fidelity term that measures the deviation from the original model behavior, and a penalty term that constrains the latency ratio to be within a tolerance margin τ.

$$\min_{\mathcal{R}} \quad \mathcal{L}(\mathcal{R}) + \lambda \cdot (\max(|\mathcal{G}(\mathcal{R}) - \alpha| - \tau, 0))^2 \tag{1}$$

(1) Representation of $\mathcal{L}(\mathcal{R})$: We use the attention score as a proxy for a token's impact on the model output. Specifically, for a prompt x with n_x tokens, we first compute a layer-wise attention score matrix $A^{(x)} \in \mathbb{R}^{L \times n_x}$, where $A^{(x)}_{i,t}$ denotes the accumulated attention score of token t at layer i. Let $\mathcal{P}^{(x)}_i(\mathcal{R})$ be all the tokens pruned after layer i by applying cascaded top-k selection to $A^x[i,:]$ [2]. We define $\mathcal{L}(\mathcal{R})$ as $\mathcal{L}(\mathcal{R}) = \frac{1}{|D|}\sum_{x\in D}\sum_{i=1}^{L}\sum_{t\in\mathcal{P}^{(x)}_i(\mathcal{R})}(A^{(x)}_{i,t})^2$ where D is the calibration dataset. $\mathcal{L}(\mathcal{R})$ measures the total importance of the tokens removed throughout the pruning schedule. Thus, minimizing $\mathcal{L}(\mathcal{R})$ encourages the pruning schedule to remove tokens with lower layer-wise importance, thereby better preserving the behavior of the original model. We further provide a sanity check of the surrogate objective in Appendix J.

(2) Representation of $\mathcal{G}(\mathcal{R})$: We approximate $\mathcal{G}(\mathcal{R})$ based on the theoretical FLOPs calculated for the average sequence length in the calibration dataset. Given a prompt with length n, the computation cost of a Transformer layer is modeled as $c_1n^2 + c_2n$, where c_1 and c_2 are architecture-dependent constants. The quadratic term corresponds to attention, while the linear term accounts for dense operations such as projections and MLPs. If a fraction r_i of tokens is pruned at layer i, the layer cost becomes $c_1((1-r_i)n)^2 + c_2((1-r_i)n)$. Thus, $\mathcal{G}(\mathcal{R}) = \frac{\sum_{i=1}^{L}(c_1(1-r_i)^2n^2+c_2(1-r_i)n)}{L(c_1n^2+c_2n)}$. A smaller $\mathcal{G}(\mathcal{R})$ implies lower prefill cost and hence higher theoretical speedup.

3.1.2 Optimization Methods

We use CMA-ES [14, 13] to optimize the surrogate objective [3]. Rather than optimizing $\mathcal{R}$ directly, we reparameterize it by local pruning variables $\mathcal{B} = \{\beta_1, \beta_2, \ldots, \beta_L\}$, where $\beta_i \in [0, 1]$ denotes the fraction of surviving tokens pruned after layer i. For instance, if $\beta_1 = 0.1$ and $\beta_2 = 0.2$, the first layer removes 10% of all tokens, and the second layer removes 20% of the remaining tokens. This corresponds to cumulative pruning ratios $r_1 = 0.1$ and $r_2 = 0.28$. Therefore, $\mathcal{B}$ and $\mathcal{R}$ are related by $\beta_i = 1 - \frac{1-r_i}{1-r_{i-1}}, r_0 = 0$.

This reparameterization is critical because the cumulative schedule $\mathcal{R}$ is not naturally suited to CMA-ES. Directly optimizing $\mathcal{R}$ requires satisfying the cascaded constraint $0 \le r_1 \le r_2 \le \cdots \le r_L < 1$, whereas CMA-ES is better matched to unconstrained or independently box-constrained variables such as $\mathcal{B}$. Since CMA-ES samples candidates from a Gaussian distribution and updates its search distribution based on the correspondence between sampled perturbations and observed fitness, applying it directly to $\mathcal{R}$ would produce many infeasible samples. Repairing these samples by sorting is inappropriate, because the evaluated schedule would no longer preserve the meaning of

[2] At layer i, top-k selection is performed only among tokens retained after layer i-1, using their scores in $A^x[i,:]$.

[3] Detailed reason is provided in Appendix A

the sampled perturbation, which can distort CMA-ES updates. In contrast, optimizing $\mathcal{B}$ guarantees feasibility by construction.

To reduce search overhead, we implement the objective computation in a tensorized manner on GPU and parallelize it across CMA-ES candidates. These optimizations, including batched evaluation, population-level parallelism, and intermediate-result caching, make the search practical on edge. Detailed implementation is provided in the Appendix I.

3.2 Online Computation

3.2.1 KQV Token Selection

At a layer, after computing the query, key, and value vectors, our goal is to retain only a subset of KQV tokens for subsequent computation. To estimate their importance online, we use the last ws query positions to attend to all key positions and summarize the resulting attention scores over the window.

We allow different KV heads to retain different token sets, since different heads often capture distinct attention patterns. To preserve a regular memory layout and balanced kernel execution, all KV heads retain the same number of tokens. Following [46, 39], we also always preserve a sink region and a recent context window ws. Therefore, the number of additional positions selected by top-k is $k = n(1 - r_i) - ws - sink_size$, where n is the prompt length.

For keys and values, the retention indices are constructed independently for each KV head. Fig. 5 shows an example of head 0; We compute attention scores from the query window to all keys in that head, aggregate the scores across the window, and apply top-k selection to determine the retained tokens (i.e., 0 and 2). The corresponding key and value tokens are then compacted into the KV cache according to these head-specific indices. For GQA, where multiple query heads share one KV head, we compute the score for each query head separately and take the maximum across the group before top-k selection.

We implement this compaction with a customized kernel, where each Cooperative Thread Array (CTA) is assigned to a single KV head. Each CTA gathers the retained KV tokens for its assigned head and writes them sequentially into a contiguous cache region. This head-level partitioning eliminates cross-CTA synchronization.

For query, in contrast, we use a shared retention index across heads. Specifically in Fig. 6, we take the position-wise maximum of the summed scores over all heads and then apply the same top-k selection. This shared query index retains any token that is important to at least one head. However, we do not compact the query vectors. We explain the rationale in the next section.

3.2.2 Boundary-Aware Sparse Attention Kernel

At runtime, each CTA computes attention between one query tile of a query head and all the KV tiles of the corresponding KV head. As shown in Fig.7, the query tile is loaded once into SRAM and reused throughout the KV scan, whereas keys and values are consumed in a streaming manner. We therefore keep query vectors in their original layout to avoid an extra gather-and-write step, and compact only the retained key and value tokens into contiguous storage to enable efficient block-wise KV traversal.

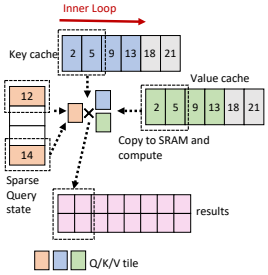

Figure 7: Sparse attention execution pipeline.

To avoid unnecessary KV scans under causal attention, we introduce a boundary-compute step for each query tile. For example, suppose the retained KV positions of one head are $[2, 5, 9, 13, 18, 21]$, and a query tile contains positions $[12, 14]$. Since a query tile can only attend to KV tokens whose original positions do not exceed the largest query position, the visible prefix is $[2, 5, 9, 13]$. The CTA therefore scans only the tiles intersecting this visible prefix and skips all later tiles containing positions 18 and 21 (gray block in Fig.7). In practice, we compute this boundary efficiently using GPU-supported binary search over the sorted retained KV indices.

Within each valid tile computation, we follow FlashAttention-style online softmax to avoid materializing attention scores in HBM, and use tensor cores for the matrix multiplications. The kernel loops

over only the boundary-valid compacted KV tiles and writes the computed outputs directly to the output buffer.

3.2.3 Execution Path Selection Mechanism

Another problem is that sparse execution is not always beneficial: for short sequences or low pruning ratios, the overhead of index construction and KV compaction can outweigh the computation saved by pruning. We therefore choose between the sparse and dense paths online before feeding into a Transformer layer.

To do so, we use lightweight polynomial models to predict the latency of the two paths and select the faster path. The sparse path is modeled as $\hat{T}_{\text{sparse}}(n, r) = \theta_0^{(s)} + \theta_1^{(s)} n + \theta_2^{(s)} nr + \theta_3^{(s)} n^2 r^2$, where the last two terms capture the computation overhead and remaining terms represent the additional overhead. The dense path is $\hat{T}_{\text{dense}}(n) = \theta_0^{(d)} + \theta_1^{(d)} n + \theta_2^{(d)} n^2$. Coefficients are fitted offline by least squares over different sequence lengths n and pruning ratios r, yielding 94% path-selection accuracy with negligible overhead.

The latency-model fitting is performed as a separate, one-time hardware profiling stage, independent of the CMA-ES schedule search. CMA-ES uses the FLOPs-based surrogate $\mathcal{G}(\mathcal{R})$, while the fitted latency models are used only for runtime path selection.

4 Experiments

4.1 Setup

Baselines. We evaluate FastPrefill against seven representative baselines: (1) SpecPrefill [26] and (2) LLMLingua [17], both of which employ a speculative model to select a subset of prompt tokens for main model inference. (3) MInference [18] and (4) DuoAttention [40], which accelerate attention computation by assigning heterogeneous attention patterns to different heads. (5) PyramidInfer [41], a KV cache compression approach that retains pivotal context layer-wise. (6) SlimInfer [28], which prunes redundant tokens at intermediate layers. (7) Original with no pruning. For all baselines, we adopt the configurations reported in their respective original papers.

Datasets and Models. We evaluate FastPrefill across two benchmarks: (1) LongBench [3], which encompasses six task categories designed to assess diverse long-context capabilities. (2) RULER [16], a synthetic benchmark that evaluates model performance by extending the "needle-in-a-haystack" test with complex tasks. We conduct experiments on Qwen2.5-7B-Instruct [32] and Llama3.1-8B-Instruct [11]. For SpecPrefill, we employ Llama3.2-1B-Instruct and Qwen2.5-1.5B as the speculative drafting models for Llama3.1-8B and Qwen2.5-7B, respectively. We adapt SpecPrefill to Qwen2.5, while excluding PyramidInfer due to OOM errors beyond 4K tokens and DuoAttention due to the failure to obtain a converged Qwen-specific head assignment.

Evaluation Details. Quality experiments are run on an 8-GPU RTX 4090 PCIe 4.0 server, while efficiency is measured on a single RTX 4090 to emulate local deployment. We implement FastPrefill with HuggingFace Transformers [37]. However, it is conceptually compatible with vLLM [21] and SGLang [47] after adapting the attention backend and KV-cache manager. We use greedy decoding for stable evaluation. Unless otherwise specified, we set $\tau = 0.01$, $\alpha = 0.5$ for Llama3.1-8B and $\alpha = 0.55$ for Qwen2.5-7B. Automatic per-query adaptation is left for future work. We update λ according to the iteration step t, with $\lambda_0 = \mathcal{L}(\mathcal{R}_{\text{init}})/0.1^2$ and $\lambda_t = \lambda_0(1 + 0.1t)$. The window/sink sizes are 16/16 for Llama3.1-8B and 8/64 for Qwen2.5-7B. The calibration set contains 390 LongBench samples and is excluded from final evaluation.

4.2 Longbench Results

Table 1 shows that FastPrefill achieves competitive overall performance. Remarkably, FastPrefill even surpasses the Full baseline on specific tasks, such as Multi-Doc QA and Synthetic tasks under the Llama3.1-8B model. This improvement may come from filtering redundant or noisy tokens during attention, leading to more focused feature extraction. However, we observe a larger degradation on Summarization where useful information is distributed across many positions. Therefore prun-

Table 1: Performance of baselines on the LongBench dataset.

Models	Methods	Single-Doc QA	Multi-Doc QA	Summarization	Few-shot Learning	Synthetic	Code Completion	Avg
Llama3.1-8B	Full	48.23	41.79	26.18	63.31	67.78	62.12	51.56
	DuoAttention	47.73	40.2	25.89	63.29	67.89	60.5	50.91
	LlmLingua (90%)	43.57	34.77	26.36	57.25	67.23	45.41	45.76
	MInference	48.13	41.59	26.04	63.76	67.24	61.25	51.34
	SlimInfer	47.13	40.68	25.89	63.25	67.68	61.17	50.97
	SpecPrefill (50%)	47.57	40.68	25.55	62.36	62.78	53.02	48.66
	SpecPrefill (70%)	47.62	41.33	26.02	63.14	63.92	55.41	49.57
	SpecPrefill (90%)	48.35	41.06	26.19	62.76	66.0	59.22	50.60
	FastPrefill	47.41	43.47	23.48	63.31	69.42	61.7	51.47
Qwen2.5-7B	Full	43.7	40.68	25.07	63.08	68.0	64.95	50.91
	LlmLingua (90%)	40.34	41.79	23.43	51.3	66.96	41.76	44.26
	MInference	46.55	41.33	23.72	62.49	66.5	65.15	50.96
	SlimInfer	46.27	40.55	23.61	62.16	68.17	63.06	50.64
	SpecPrefill (50%)	43.23	41.36	24.27	67.32	59.64	64.71	50.09
	SpecPrefill (70%)	43.00	39.76	24.69	68.30	61.32	63.85	50.15
	SpecPrefill (90%)	43.62	40.25	24.70	69.33	61.56	65.20	50.78
	FastPrefill	45.39	40.5	21.77	62.03	67.83	63.98	50.25

Table 2: Performance of baselines on the RULER dataset.

	Task Length	Retrieval	Multi-hop Tracking	Aggregation	QA	Avg
Origin	4K	99.84	91.85	92.47	68.90	92.72
	8K	98.91	89.91	92.44	65.90	91.45
	16K	93.86	89.09	76.06	64.40	85.86
	32K	97.42	87.52	40.60	61.70	81.66
FastPrefill	4K	98.71	90.94	80.21	68.10	89.96
	8K	97.99	89.22	84.12	65.70	89.54
	16K	92.50	89.59	73.38	64.10	84.74
	32K	97.21	88.87	42.08	61.70	81.98

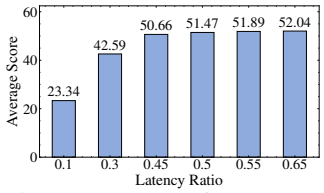

Figure 8: Impact of latency ratio on LongBench score.

ing tokens with low attention scores may weaken the global information integration for summary generation.

Moreover, baselines such as LLMLingua or SpecPrefill [4] suffer from noticeable performance degradation due to weak auxiliary model extraction capabilities. This indicates that FastPrefill can understand long-context with local attention scores. Although MInference and SlimInfer achieve similar performance, FastPrefill demonstrates superior efficiency. We exclude PyramidInfer from the quality comparison because it frequently runs out of memory in local deployment (Fig. 9), making its quality measurement less meaningful under our evaluation setting.

4.3 RULER Results

We further evaluate FastPrefill on RULER under varying context lengths. Table 2 shows that FastPrefill achieves performance close to the original model across different context lengths. For retrieval, multi-hop tracking, and QA, the scores remain stable, indicating that progressive pruning can preserve the critical evidence tokens required by many long-context tasks. The main degradation appears on aggregation tasks. This is expected because aggregation tasks distribute information across many positions, making frequency counting (FWE) and common-token extraction (CWE) particularly vulnerable to pruning. Nevertheless, the overall performance remains close to the original model, and FastPrefill even slightly improves the average score at 32K. These results suggest that FastPrefill provides a favorable quality-efficiency trade-off under the default latency ratio, while aggregation-heavy workloads may benefit from a more conservative pruning setting.

4.4 Robustness Results on Domain Shift

We evaluate calibration robustness by independently optimizing pruning schedules on three out-of-distribution datasets: L-Eval [2], BookSum [20], and Tau2 [4]. As shown in Table 3, the LongBench-calibrated schedule achieves an average LongBench score of 51.47, while the schedules calibrated on BookSum, Tau2, and L-Eval achieve 50.97, 50.78, and 50.35, respectively, corresponding to 97.8%–99.0% retention of the LongBench-calibrated performance. Performance on natural-language QA and summarization remains particularly stable, while larger variations are mainly concentrated in the synthetic and code categories. These results demonstrate that the learned pruning schedules are

[4]The percentage is the token retention ratio

Table 3: Robustness to calibration distribution shifts on LongBench.

Calibration Set	Single-Doc QA	Multi-Doc QA	Summarization	Few-shot Learning	Synthetic	Code Completion	Avg.	Δ Avg.
LongBench	47.41	43.47	23.48	63.31	69.42	61.70	51.47	–
BookSum	48.03	42.74	24.77	63.67	66.66	59.95	50.97	-0.50
Tau2	48.07	42.60	24.46	63.52	66.68	59.37	50.78	-0.69
L-Eval	47.10	42.78	23.64	62.32	67.21	59.05	50.35	-1.12

relatively robust to changes in the calibration distribution, although some task-specific sensitivity remains.

4.5 Efficiency Results

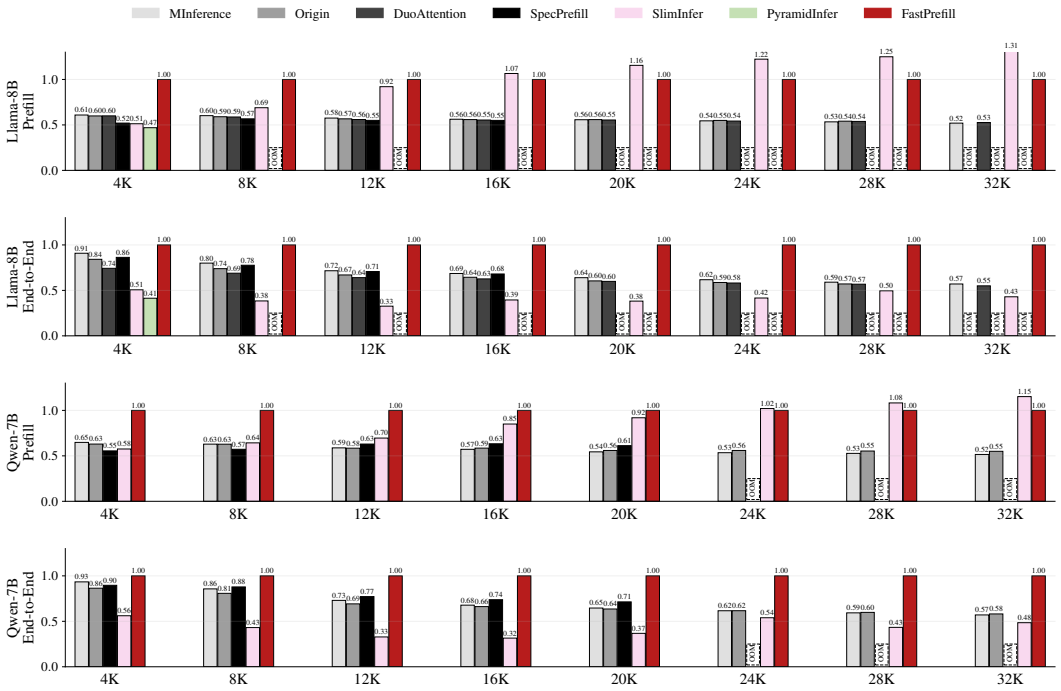

Figure 9: Speedup results across Llama3.1-8B and Qwen2.5-7B under various prompt lengths. FastPrefill achieves higher prefill speedup than attention-only and auxiliary-model-assisted baselines, and outperforms SlimInfer (recovery-based method) with lower prefill latency on short prompts and lower end-to-end latency across all prompt lengths.

Experimental Setup. We use synthetic data as inputs. To ensure comparable quality to the original model, SpecPrefill employs a 90% retention ratio on Llama and 70% on Qwen. LLMLingua is excluded due to significant quality degradation. We report TTFT and end-to-end latency for 32 generated tokens.

Results. First, in Fig. 9, FastPrefill consistently outperforms MInference on both Llama and Qwen, and DuoAttention on Llama, across all evaluated prompt lengths. This indicates that optimizing attention alone is insufficient for substantial long-context prefill acceleration, since the FFN cost remains unchanged. Furthermore, PyramidInfer encounters OOM after 4K tokens due to the heavy memory footprint of computing full attention scores. FastPrefill resolves this by recomputing attention within query windows and designing a custom attention operator.

Second, the comparison with SlimInfer underscores the necessity of a recovery-free formulation. Although SlimInfer's aggressive pruning achieves faster raw prefill for longer sequences, this advantage does not translate into practical efficiency. Specifically, its end-to-end speedup decreases as the decoding length grows (e.g., 32 generated tokens), since decoding-phase retrieval repeatedly introduces synchronization and I/O overheads. In contrast, FastPrefill consistently maintains lower

Table 4: Latency and peak VRAM of different attention implementations. Each entry reports (latency in ms, peak memory in MB).

Methods	(4K,0.25)	(4K,0.5)	(16K,0.25)	(16K,0.5)
Dense	(0.99, 32.5)	(1.03, 32)	(13.96, 130)	(13.91, 130)
Naive Pruning	(8.63, 1600)	(8.79, 1600)	OOM	OOM
Gather+Causal	(2.26, 436)	(7.92, 840)	OOM	OOM
FastPrefill	(0.72, 43)	(0.91, 48)	(3.14, 192)	(6.84, 192)

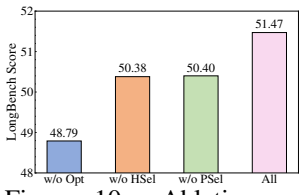

Figure 10: Ablation results. Opt: Optimizer, HSel: head-specific selection, PSel: path selection.

end-to-end latency by avoiding such recovery costs. This demonstrates that FastPrefill can trade off inference quality and efficiency with only prefill pruning.

4.6 End-to-End Speedup Breakdown

For Llama3.1-8B, FFN reduction provides the largest latency saving by directly reducing its large, regular GEMMs, while sparse attention gains are smaller due to fragmented execution. At 8K, 16K, and 24K, attention accounts for 25.6%, 24.6%, and 21.5% of the latency savings, respectively. FFN reduction accounts for 65.0%, 52.5%, and 50.6%, and path selection contributes 3.3%, 22.3%, and 27.7%. We exclude a separate breakdown of SlimInfer's decoding-time recovery overhead because it is integrated into sparse attention and overlaps with attention computation. Isolating it would require restructuring the pipeline, altering this overlap and making the measurement unfair.

4.7 Sensitivity to Latency Ratio

This experiment studies how the latency ratio controls the quality-efficiency trade-off on LongBench. Fig. 8 shows that as the latency ratio increases from 0.1 to 0.65, the average score improves from 23.3 to 52.04, indicating that aggressive pruning hurts quality. However, the improvement is highly non-uniform and quickly saturates after 0.5. Therefore, we choose 0.5 as the default latency ratio in our experiments. It lies near the knee point of the curve, achieves a strong LongBench score of 51.47, and avoids the diminishing returns of more conservative pruning. In practical deployment, this ratio is selected to target general-purpose workloads. For information-sensitive tasks such as summarization, code completion or aggregation, users may choose a more conservative latency ratio to avoid quality degradation. We provide detailed results in Appendix B and other sensitivity results in Appendix C.

4.8 Ablation Study

We evaluate three key modules of FastPrefill by removing them individually. (1) We disable the offline optimizer, denoted as "w/o Opt", and replace the optimized pruning schedule with a heuristic exponential schedule that achieves the same wall-clock speedup as FastPrefill. (2) We remove the head-specific KV selection strategy, denoted as "w/o HSel", by forcing all KV heads to share the same retained token indices. (3) We disable runtime path selection, denoted as "w/o PSel", and always execute the sparse path.

Fig. 10 shows that removing the offline optimizer causes the largest drop, from 51.47 to 48.79, indicating that the optimizer is most important for preserving quality. This suggests that a simple hand-crafted schedule is inferior to the data-driven optimized schedule. Removing head-specific KV selection reduces the score to 50.38, suggesting that keeping heterogeneous attention patterns across KV heads can maintain head-specific information. Disabling path selection also leads to a performance drop, because pruning may cause information loss.

4.9 Overhead of Sparse Attention Execution

Table 4 compares the non-projection part of attention execution across dense attention, naive pruning with full logical masking, gathered KQV with a compressed causal mask, and FastPrefill. The measured latency includes both writing the retained KV states into the cache and performing the attention computation, while excluding QKV/O projection costs. The results validate our motivation. Dense attention brings no performance gain because it still performs full attention computation, while

a simple eager implementation is costly and OOM-prone. In contrast, despite the cost of recomputing attention scores, FastPrefill consistently improves latency especially when the sequence length is large, because of higher GPU utilization (4.45× speedup). FastPrefill incurs extra memory overhead due to intermediate scores and auxiliary variables. This overhead is acceptable compared with several GBs of total memory usage in long-context inference. Moreover, the memory usage across different sparsity ratios under the same sequence length is not strictly monotonic because of PyTorch [31]'s memory pool management mechanism [12].

4.10 Offline Optimizer Cost

We compare the runtime of the optimized offline search implementation against a non-optimized version. The optimizer is executed on the calibration dataset with 390 samples. The optimized implementation reduces the per-iteration cost from 1.01 s to 0.09 s for Llama3.1-8B, yielding an 11.2× speedup. As a result, offline schedule search becomes practical in our setting: 600 iterations are typically sufficient for convergence and finish in about 60s.

5 Related Work

Layer-wise Token Pruning. Several studies accelerate prefill by progressively reducing tokens across Transformer layers. PyramidInfer [41] exploits an empirical power-law pruning schedule and prunes hidden states before each layer, but its heuristic design yields limited speedup in practice. LazyLLM [9] and SlimInfer [28] adopt more aggressive hand-crafted schedules, such as retaining only a fixed number of tokens after certain layers, to minimize nominal prefill cost. However, to preserve quality, they rely on recovering pruned tokens during decoding. This makes their latency gains less direct in practical single-GPU deployment, where recovery can introduce additional runtime overhead.

Compression Techniques. Current compression approaches typically fall into KV cache compression and parameter compression. Eviction-based KV methods [10, 24, 5, 46, 27, 1, 43] maintain a small subset of critical KV caches. Nevertheless, since they conduct token identification post-prefill, they inherently cannot reduce the TTFT and accelerate the prefill phase. Other techniques like KV quantization [6, 19, 42, 8, 15, 43] are orthogonal to our research. Furthermore, parameter compression, including weight pruning [35, 45, 38] and quantization [25, 30, 33, 43], primarily minimizes the static memory footprint and alleviates memory-bandwidth bottlenecks during the decoding phase. They are also orthogonal to our research.

6 Conclusion

We present FastPrefill, an algorithm–system co-design framework for recovery-free prefill acceleration. Instead of relying on heuristic pruning rules or token recovery, FastPrefill jointly optimizes layer-wise pruning schedules and runtime sparse execution. Its offline optimizer searches for quality-preserving schedules under a target latency constraint, while its online pipeline realizes head-specific dynamic pruning through efficient sparse attention. Comprehensive experiments show that FastPrefill achieves substantial prefill speedup while maintaining inference quality, demonstrating that pruning can be both effective and system-efficient when schedule optimization and execution are designed together.

Acknowledgements and Disclosures

This work was supported by the National Natural Science Foundation of China under Grant No. W2531047. The authors declare no competing interests. We thank the anonymous reviewers and the Area Chair for their constructive feedback.

References

[1] M. Adnan, A. Arunkumar, G. Jain, P. J. Nair, I. Soloveychik, and P. Kamath. Keyformer: Kv cache reduction through key tokens selection for efficient generative inference. *Proceedings of Machine Learning and Systems*, 6:114–127, 2024.

[2] C. An, S. Gong, M. Zhong, X. Zhao, M. Li, J. Zhang, L. Kong, and X. Qiu. L-eval: Instituting standardized evaluation for long context language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14388–14411, 2024.

[3] Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang, Z. Du, X. Liu, A. Zeng, L. Hou, et al. 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, 2024.

[4] V. Barres, H. Dong, S. Ray, X. Si, and K. R. Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. In *Forty-third International Conference on Machine Learning*, 2026.

[5] Y. Chen, G. Wang, J. Shang, S. Cui, Z. Zhang, T. Liu, S. Wang, Y. Sun, D. Yu, and H. Wu. Nacl: A general and effective kv cache eviction framework for llm at inference time. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7913–7926, 2024.

[6] W. Cheng, S. Dong, J. Qin, and W. Wang. Qaq: Quality adaptive quantization for llm kv cache. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2542–2550, 2025.

[7] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35: 16344–16359, 2022.

[8] H. Dong, X. Yang, Z. Zhang, Z. Wang, Y. Chi, and B. Chen. Get more with less: Synthesizing recurrence with kv cache compression for efficient llm inference. *arXiv preprint arXiv:2402.09398*, 2024.

[9] Q. Fu, M. Cho, T. Merth, S. Mehta, M. Rastegari, and M. Najibi. Lazyllm: Dynamic token pruning for efficient long context llm inference. *arXiv preprint arXiv:2407.14057*, 2024.

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

[11] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[12] C. Guo, R. Zhang, J. Xu, J. Leng, Z. Liu, Z. Huang, M. Guo, H. Wu, S. Zhao, J. Zhao, et al. Gmlake: Efficient and transparent gpu memory defragmentation for large-scale dnn training with virtual memory stitching. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 450–466, 2024.

[13] N. Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[14] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[15] C. Hooper, S. Kim, H. Mohammadzadeh, M. W. Mahoney, Y. S. Shao, K. Keutzer, and A. Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

[16] C.-P. Hsieh, S. Sun, S. Kriman, S. Acharya, D. Rekesh, F. Jia, Y. Zhang, and B. Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

[17] H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, and L. Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 13358–13376, 2023.

[18] H. Jiang, Y. Li, C. Zhang, Q. Wu, X. Luo, S. Ahn, Z. Han, A. H. Abdi, D. Li, C.-Y. 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.

[19] H. Kang, Q. Zhang, S. Kundu, G. Jeong, Z. Liu, T. Krishna, and T. Zhao. Gear: An efficient kv cache compression recipe for near-lossless generative inference of llm. *arXiv preprint arXiv:2403.05527*, 2024.

[20] W. Kryściński, N. Rajani, D. Agarwal, C. Xiong, and D. Radev. Booksum: A collection of datasets for long-form narrative summarization. 2021.

[21] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. 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.

[22] M. Labonne and S. Moran. Spam-t5: Benchmarking large language models for few-shot email spam detection, 2023. URL `https://arxiv.org/abs/2304.01238`.

[23] X. Lan, Z. Han, Y. Cheng, L. Sheng, J. Feng, C. Gao, and Y. Li. Depression detection on social media with large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 2155–2171, 2025.

[24] Y. Li, Y. Huang, B. Yang, B. Venkitesh, A. Locatelli, H. Ye, T. Cai, P. Lewis, and D. Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

[25] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6:87–100, 2024.

[26] J. Liu, B. Chen, and C. Zhang. Speculative prefill: Turbocharging TTFT with lightweight and training-free token importance estimation. In *Forty-second International Conference on Machine Learning*, 2025.

[27] Z. Liu, A. Desai, F. Liao, W. Wang, V. Xie, Z. Xu, A. Kyrillidis, and A. Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364, 2023.

[28] L. Long, R. Yang, Y. Huang, D. Hui, A. Zhou, and J. Yang. Sliminfer: Accelerating long-context llm inference via dynamic token pruning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 32284–32292, 2026.

[29] OpenAI. Gpt-4 technical report, 2024. URL `https://arxiv.org/abs/2303.08774`.

[30] Y. Park, J. Hyun, H. Kim, and J. W. Lee. Decdec: A systems approach to advancing low-bit llm quantization. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, pages 803–819, 2025.

[31] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[32] Qwen, A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Tang, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu. Qwen2.5 Technical Report, Jan. 2025.

[33] W. Shao, M. Chen, Z. Zhang, P. Xu, L. Zhao, Z. Li, K. Zhang, P. Gao, Y. Qiao, and P. Luo. Omniquant: Omnidirectionally calibrated quantization for large language models. *arXiv preprint arXiv:2308.13137*, 2023.

[34] Y. Song, Z. Mi, H. Xie, and H. Chen. Powerinfer: Fast large language model serving with a consumer-grade gpu. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pages 590–606, 2024.

[35] M. Sun, Z. Liu, A. Bair, and J. Z. Kolter. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[37] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[38] H. Xia, Z. Zheng, Y. Li, D. Zhuang, Z. Zhou, X. Qiu, Y. Li, W. Lin, and S. L. Song. Flash-llm: Enabling cost-effective and highly-efficient large generative model inference with unstructured sparsity. *Proc. VLDB Endow.*, 17(2):211–224, Oct. 2023. ISSN 2150-8097.

[39] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis. Efficient streaming language models with attention sinks, 2024. URL `https://arxiv.org/abs/2309.17453`.

[40] G. Xiao, J. Tang, J. Zuo, junxian guo, S. Yang, H. Tang, Y. Fu, and S. Han. Duoattention: Efficient long-context LLM inference with retrieval and streaming heads. In *The Thirteenth International Conference on Learning Representations*, 2025.

[41] D. Yang, X. Han, Y. Gao, Y. Hu, S. Zhang, and H. Zhao. Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3258–3270, 2024.

[42] J. Y. Yang, B. Kim, J. Bae, B. Kwon, G. Park, E. Yang, S. J. Kwon, and D. Lee. No token left behind: Reliable kv cache compression via importance-aware mixed precision quantization. *arXiv preprint arXiv:2402.18096*, 2024.

[43] S. Yang, J. Guo, H. Tang, Q. Hu, G. Xiao, J. Tang, Y. Lin, Z. Liu, Y. Lu, and S. Han. Lserve: Efficient long-sequence llm serving with unified sparse attention. *Proceedings of Machine Learning and Systems*, 7, 2025.

[44] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models, 2023. URL `https://arxiv.org/abs/2210.03629`.

[45] Y. Zhang, L. Zhao, M. Lin, S. Yunyun, Y. Yao, X. Han, J. Tanner, S. Liu, and R. Ji. Dynamic sparse no training: Training-free fine-tuning for sparse LLMs. In *The Twelfth International Conference on Learning Representations*, 2024.

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

[47] L. Zheng, L. Yin, Z. Xie, C. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

A Discussion

About quality loss. Quality loss is a common limitation of pruning methods, because discarded tokens cannot contribute to the following decoding process. Our contribution is not to make pruning lossless, but to reduce this loss while preserving the system benefit of recovery-free execution. Specifically, FastPrefill uses target-model attention signals, an offline layer-wise schedule optimizer, head-specific KV selection, and conservative latency-ratio control to achieve a better quality-efficiency trade-off without decoding-time token recovery. The remaining risk is more visible in aggregation-heavy or summarization tasks, where useful information may be distributed across many individually low-salience tokens. In such cases, a more conservative latency ratio or task-adaptive pruning policy can be used to further reduce quality degradation. We leave the selection of latency ratio to future work.

Integrating with industry engines. FastPrefill is conceptually compatible with production serving systems such as vLLM [21] and SGLang [47], but it is not a plug-and-play Python-level wrapper. Integration would mainly involve three components. First, the prefill attention backend needs to be extended to support FastPrefill's boundary-aware sparse attention kernel. Second, the KV-cache manager needs to store compacted KV states together with retained-token metadata, including the original token positions and per-head retained KV indices. Third, the runtime scheduler needs to pass the layer-wise pruning schedule and retained-token metadata to the attention backend during prefill. We leave the integration for future work.

Choice of CMA-ES for schedule optimization. We select CMA-ES by jointly considering search efficiency and the structure of our layer-wise schedule optimization problem. (1) Population-based methods, including differential evolution, genetic algorithms, particle swarm optimization, and CMA-ES, support global exploration and parallel candidate evaluation. Among them, CMA-ES additionally adapts the covariance of its sampling distribution, allowing it to capture correlations among pruning decisions at different layers. Its population can also be evaluated efficiently using our optimized batched GPU implementation described in Appendix I. (2) Exhaustive and non-adaptive methods are less suitable for our high-dimensional search space. For a 32-layer model, even discretizing each layer-wise pruning variable into three candidate values yields $3^{32} \approx 1.85 \times 10^{15}$ possible schedules, making grid search infeasible. Random and quasi-random search do not use feedback from previous evaluations to focus on promising regions. (3) Direct-search methods, such as coordinate search, pattern search, Nelder–Mead, and Powell's method, can handle non-differentiable objectives but do not explicitly learn the full cross-variable covariance structure, which is important for coordinating pruning decisions across layers. (4) Standard Gaussian-process Bayesian optimization faces challenges in our high-dimensional search space (28 variables for Qwen and 32 for Llama), while the discrete changes induced by cascaded top-k selection limit the effectiveness of conventional smooth kernels. (5) Zeroth-order methods such as finite differences and SPSA may produce unreliable gradient estimates, since small changes in pruning ratios can either leave the selected tokens unchanged or cause abrupt changes in token selection.

B More Sensitivity Results to Latency Ratio

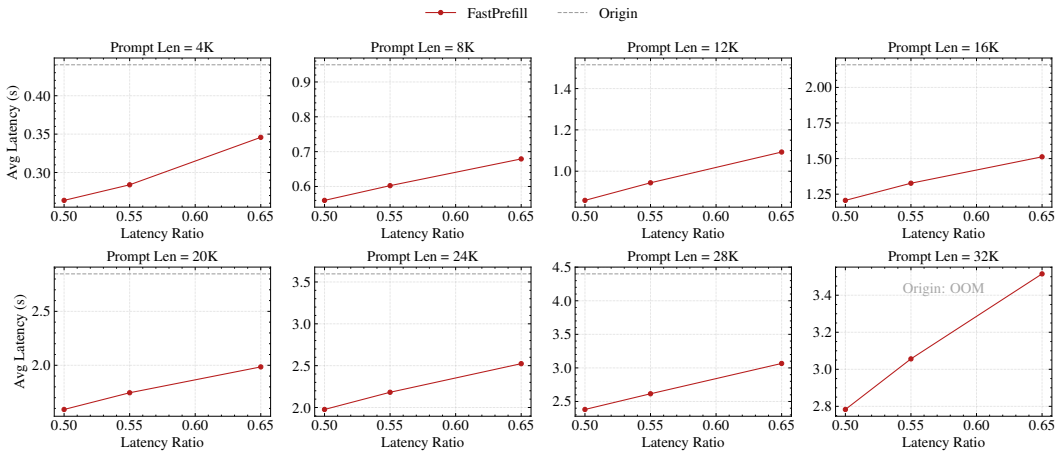

Figure 11: TTFT variance in Llama3.1-8B with different latency ratio under different prompt length.

We further study how the latency ratio affects both model quality and real TTFT latency on Llama3.1-8B. As shown in Table. 5, increasing the latency ratio consistently improves the LongBench score. The average score increases from 23.34 at a highly aggressive ratio of 0.1 to 51.47 at 0.5, and further reaches 52.06 at 0.65. The improvement becomes much smaller after 0.5, indicating that 0.5 is close to the knee point of the quality-efficiency curve. This also explains our choice of 0.5 as the default setting for general workloads. Meanwhile, some information-dense tasks, such as Summarization, benefit from a more conservative latency ratio, because more prompt tokens are preserved for global information aggregation. Fig. 11 shows the corresponding wall-clock TTFT under different prompt lengths. The latency increases monotonically as the latency ratio becomes larger, which confirms that the latency ratio effectively controls real system speed, not only the theoretical cost. Importantly,

Table 5: Sensitivity analysis of latency ratio α with Llama3.1-8B.

α	Single-Doc	Multi-Doc	Summarization	Few-shot	Synthetic	Code	Avg
0.1	14.92	19.11	16.63	31.23	7.0	51.16	23.34
0.3	32.6	35.77	19.85	47.35	63.85	56.13	42.59
0.45	47.15	42.02	23.72	60.4	68.64	62.01	50.66
0.5	47.41	43.47	23.48	63.31	69.42	61.7	51.47
0.55	48.38	43.85	24.69	63.36	68.74	62.34	51.89
0.65	48.68	43.91	25.3	64.11	68.22	62.13	52.06
Full	48.23	41.79	26.18	63.31	67.78	62.12	51.56

Table 6: Sensitivity analysis of window size and sink size on LongBench with Llama3.1-8B at $\alpha = 0.5$. The best result in each column is highlighted.

Window	Sink	Single-Doc	Multi-Doc	Summarization	Few-shot	Synthetic	Code	Avg
8	8	48.13	42.36	25.11	**63.74**	67.66	60.67	51.28
16	8	48.07	42.29	24.98	63.28	67.20	61.50	51.22
32	8	47.82	42.02	24.66	62.96	66.54	61.87	50.98
64	8	47.33	42.05	24.42	62.51	67.96	62.02	51.05
8	16	**48.17**	42.54	25.14	63.64	66.93	60.55	51.16
16	16	47.41	**43.47**	23.48	63.31	**69.42**	61.70	**51.47**
32	16	47.72	42.45	24.75	63.24	67.21	61.48	51.14
64	16	47.51	41.94	24.40	62.73	67.46	62.03	51.01
8	32	48.03	42.39	25.06	63.60	67.09	60.69	51.14
16	32	48.08	42.71	**25.16**	63.02	67.34	61.76	51.34
32	32	47.81	42.32	24.75	63.12	66.88	61.66	51.09
64	32	47.49	42.11	24.37	62.56	67.38	61.89	50.96
8	64	48.09	42.72	25.02	63.53	66.80	60.59	51.12
16	64	48.07	42.52	25.03	63.50	67.29	61.49	51.32
32	64	47.82	41.95	24.77	62.92	67.04	62.05	51.09
64	64	47.47	41.83	24.27	62.93	67.16	**62.20**	50.98

even with the conservative ratio of 0.65, FastPrefill still remains faster than the original dense prefill for all non-OOM settings, while the original model runs out of memory at 32K tokens. These results show that the latency ratio serves as a practical deployment knob: users can choose a smaller ratio for latency-sensitive scenarios and a larger ratio for quality-sensitive or information-dense tasks.

C Sensitivity to Window Size and Sink Size

The sensitivity results in Table. 6 show that FastPrefill is robust to window size and sink size at the benchmark level. Across all tested configurations, the average LongBench score only varies from 50.96 to 51.47. However, the best configuration differs across task categories. For example, with Sink = 8, reducing the window size from 64 to 8 improves Single-Doc QA from 47.33 to 48.13 and Few-shot Learning from 62.51 to 63.74, while Code Completion improves from 60.67 to 62.02 when the window size increases from 8 to 64. These results suggest that a single global configuration is sufficient for stable overall performance, while task-adaptive hyperparameter selection may further improve category-level quality. Specifically, smaller or moderate windows are generally sufficient for QA and few-shot tasks, while code completion benefits from a larger window.

D Static Batch Experiments

Table 7 shows that FastPrefill consistently achieves the highest normalized throughput across all evaluated static batching configurations, demonstrating its effectiveness beyond batch-size-one inference.

E Sensitivity to Calibration Dataset Size and Composition

We use Llama-3.1-8B with the same optimizer settings and a target latency ratio of 0.5. To study calibration size, we compare the full 390-sample set (average length: 9,670 tokens) with a random

Table 7: Normalized throughput comparison under static batching on Llama3.1-8B. All results are normalized to FastPrefill (1.00×). Higher is better.

Batch Configuration	Stage	FastPrefill	SpecPrefill	DuoAttention	MInference	Origin
2×12K	Prefill	1.00×	0.76×	0.57×	0.44×	0.68×
	Decode	1.00×	0.76×	0.69×	0.44×	0.72×
4×6K	Prefill	1.00×	0.72×	0.58×	0.44×	0.69×
	Decode	1.00×	0.72×	0.58×	0.44×	0.69×
8×3K	Prefill	1.00×	0.66×	0.58×	0.42×	0.70×
	Decode	1.00×	0.66×	0.58×	0.42×	0.70×

Table 8: Sensitivity to calibration set size and prompt-length distribution on LongBench.

Calibration Set	Single-Doc QA	Multi-Doc QA	Summarization	Few-shot Learning	Synthetic	Code Completion	Avg.
Full (390)	47.41	43.47	23.48	63.31	69.42	61.70	51.47
Random (48)	47.94	42.20	23.04	62.61	66.28	59.93	50.33
Narrow (48)	48.06	43.09	23.17	62.84	65.73	60.09	50.50
Bimodal (48)	48.35	42.76	23.21	62.90	67.07	59.92	50.70

48-sample subset (9,566 tokens). To isolate prompt-length distribution, we construct two 48-sample sets with matched dataset composition and nearly identical mean lengths, but different standard deviations: $9,639 \pm 5,706$ tokens for bimodal and $9,647 \pm 2,229$ tokens for narrow.

As shown in Table. 8, reducing the calibration set causes a modest performance drop, while changing the length distribution produces only a 0.20-point difference. Together with the cross-domain results reported, these results indicate limited sensitivity to calibration size, composition, and prompt-length distribution.

F Analysis of Learned Pruning Schedules

FastPrefill learns one global pruning schedule for each model from the calibration set, and the pruning schedule itself does not vary across prompt lengths or tasks at inference time. Table. 9 compares optimized schedules with exponential heuristics. Ret.@x% denotes cumulative retention after x% of layers, computed over a subset of prompts; "Main layers" are the three layers with the largest absolute token removal. Optimized schedules delay pruning to the middle and later layers, while exponential schedules prune early. The exact pruning locations are model-dependent. We hypothesize that early layers require broader token coverage to build stable contextual representations, while later layers tolerate more aggressive pruning.

G Algorithm Details

As shown in Alg. 1, the boundary-aware sparse kernel is implemented at the granularity of query tiles. For each query tile, we first determine the maximum original query position within the tile (line 9). Because the retained KV indices are sorted by their original positions, the number of KV tokens visible to this query tile can be obtained by an upper-bound search on the retained KV index array (line 10). This gives the exact visible KV length for the tile. We then round this length up to the nearest multiple of the KV tile size, so that the Triton kernel can iterate over KV tiles with a static block size (line 11). Therefore, the loop bound used by the CTA is not the full retained KV length, but the tile-specific boundary determined by causal visibility.

Inside the sparse kernel, each CTA computes the attention output for one query tile of one query head. Under grouped-query attention, the query head is mapped to its corresponding KV head, and the CTA scans the compacted KV tiles of that KV head. The query tile is loaded once and reused across the KV loop, while keys and values are loaded in a streaming manner from the compacted buffers. For each KV tile, the kernel also loads the original physical KV positions and applies a fine-grained causal mask by comparing them with the physical query positions. This is necessary because the loop

Table 9: Comparison of optimized and exponential pruning schedules across models.

Model	Schedule	Ret.@25%	Ret.@50%	Ret.@75%	Final Ret.	Main Layers
Llama3.1-8B	Optimized	100.0%	65.4%	7.5%	0.3%	14, 18, 20
	Exponential	64.2%	38.7%	23.3%	14.0%	2, 3, 4
Qwen2.5-7B	Optimized	100.0%	55.1%	18.1%	1.8%	11, 16, 18
	Exponential	64.5%	38.7%	23.2%	13.9%	2, 3, 4

boundary only removes completely invisible KV tiles, while the last valid KV tile may still contain positions that are larger than some queries in the current query tile.

The online softmax update follows the same principle as FlashAttention (line 17, 22–26). The kernel maintains, for each query row, the running maximum score, the running normalization term, and the accumulated output. After computing the score matrix for a new KV tile (line 20), the row-wise maximum is updated, the previous accumulator is rescaled accordingly, and the contribution from the current value tile is added. This allows the kernel to compute attention over a sequence of KV tiles without materializing the full attention matrix in HBM. After all KV tiles up to the precomputed boundary have been processed, the accumulated output is normalized by the final normalization term and written back to global memory.

H Path Latency Model Profiling Details

For each model, we evaluate sequence lengths n from 1,024 to 64,512 with a step of 2,048 and token-retention ratios r from 0.10 to 0.90 with a step of 0.05, resulting in 544 profiling configurations. For each configuration, we perform three warm-up iterations followed by ten timed iterations using CUDA events and record the mean latency. The dense-path measurement includes standard dense attention execution, while the sparse-path measurement includes online score computation, retained-index construction, KV compaction, and boundary-aware sparse attention. Operations shared by both paths, such as QKV/O projections and FFN, are excluded. Collecting the profiling data for the dense and sparse paths takes approximately 16.2 and 13.3 minutes, respectively, for Llama3.1-8B, and 15.4 and 12.2 minutes for Qwen2.5-7B. After data collection, we fit the polynomial coefficients using ordinary least squares implemented with `numpy.linalg.lstsq`. The regression takes approximately 0.0004 s for the sparse path and 0.0002 s for the dense path on both models.

I Efficient Computation of the Offline Objective

The offline search repeatedly evaluates the surrogate objective for many candidate pruning schedules. Although this search is performed before deployment, a naive implementation can still be expensive, because each CMA-ES iteration evaluates a population of candidates, and each candidate requires scanning the calibration set across all layers. This appendix section describes how we accelerate the objective evaluation while preserving the same pruning semantics as the original formulation.

Naive evaluation. For each calibration prompt, the naive implementation processes the layers sequentially. Given a cumulative pruning schedule $\mathcal{R}$, it first computes how many tokens should have been removed by the current layer. The newly pruned tokens are then selected from the tokens that are still active, according to their layer-wise scores. Once a token is pruned, it remains pruned in all subsequent layers. The fidelity loss $\mathcal{L}$ is accumulated by summing the squared importance scores of all tokens that have already been removed at each layer.

This procedure directly follows the definition of the objective, but it has two sources of overhead. First, it repeatedly traverses samples and layers in Python, which becomes inefficient when the calibration set is large. Second, the same process must be repeated for every candidate schedule sampled by CMA-ES. Since CMA-ES evaluates a population of schedules at each iteration, evaluating candidates one by one leads to substantial redundant work.

Batched dataset representation. We first remove repeated data-loading overhead by materializing the calibration scores into GPU tensors. Each sample is represented as a matrix whose rows correspond

Algorithm 1: Attention Computation

Input: Q, K, V; retained query indices I_q; retained KV indices I_{kv}; tile sizes B_M, B_N
Output: Attention output O

Online path selection:
$\hat{T}_{\text{sparse}}, \hat{T}_{\text{dense}} \leftarrow \textsc{EstimateTime}(\theta, r, n)$
if $\hat{T}_{\text{sparse}} \geq \hat{T}_{\text{dense}}$ **then**
  return $\textsc{DenseAttention}(Q, K, V)$
else
  Construct retained indices I_q and I_{kv}
  Compact K, V into contiguous buffers K_c, V_c
  foreach *query tile* i **do**
    $q_{\max} \leftarrow \max I_q[iB_M : (i+1)B_M]$
    $L_i \leftarrow \textsc{UpperBound}(I_{kv}, q_{\max})$
    $N^i_{\text{end}} \leftarrow \lceil L_i / B_N \rceil \, B_N$
  foreach *CTA in parallel* **do**
    `// Each CTA handles one query tile of one query head and the corresponding KV head`
    $O_i \leftarrow \textsc{SparseKernel}(Q, K_c, V_c, I_q, I_{kv}, i, N^i_{\text{end}})$

Function `SparseKernel`$(Q, K_c, V_c, I_q, I_{kv}, i, N_{\text{end}})$
  Load query tile Q_i and its physical positions P_q
  Initialize $m \leftarrow -\infty$, $\ell \leftarrow 0$, $O_i \leftarrow 0$
  for $j = 0$ **to** $N_{\text{end}} - 1$ **by** B_N **do**
    Load compacted KV tile K_j, V_j and physical positions P_k
    $S \leftarrow Q_i K_j^\top / \sqrt{d}$
    Apply causal mask: $S_{ab} \leftarrow -\infty$ if $P_q[a] < P_k[b]$
    `//` a `and` b `is row index and column index`
    $m_{\text{new}} \leftarrow \max(m, \max_b S_{:,b})$
    $P \leftarrow \exp(S - m_{\text{new}})$
    $\ell_{\text{new}} \leftarrow \exp(m - m_{\text{new}})\ell + \sum_b P_{:,b}$
    $O_i \leftarrow \exp(m - m_{\text{new}})O_i + PV_j$
    $m \leftarrow m_{\text{new}}, \ell \leftarrow \ell_{\text{new}}$
  $O_i \leftarrow O_i / \ell$
  return O_i

to layers and whose columns correspond to token positions. Since prompts may have different lengths, we sort samples by length and pack samples with similar lengths into the same batch. Within each batch, shorter samples are padded to the maximum length, and a binary mask is used to prevent padded positions from participating in token selection or loss computation. This batching strategy reduces padding waste while allowing the objective to be evaluated by tensor operations on GPU.

Suffix-sum loss caching. The key observation is that the loss contribution of a token can be computed once when the token is first pruned. Suppose token t is newly pruned at layer i. Because pruning is cascaded, this token is absent from all later layers. Therefore, its total contribution to the fidelity loss is not only its score at layer i, but the sum of its squared scores from layer i to the final layer. We precompute this quantity for every sample, layer, and token:

$$S^{(x)}_{i,t} = \sum_{\ell=i}^{L} \left(A^{(x)}_{\ell,t} \right)^2 . \tag{2}$$

Then, when token t is newly pruned at layer i, we directly add $S^{(x)}_{i,t}$ to the loss. This avoids repeatedly summing over all previously pruned tokens at every later layer. As a result, the loss accumulation becomes an event-based computation: only newly pruned tokens contribute new terms, and their future contributions are already cached.

Vectorized token selection. For each candidate schedule, we convert the cumulative pruning ratios into the number of tokens that should be pruned by each layer. The difference between two adjacent cumulative counts gives the number of newly pruned tokens at that layer. We maintain a Boolean pruning mask for each sample, which records whether a token has already been removed. At each layer, we mask out both padded tokens and previously pruned tokens, and use a batched top-k operation to select the newly pruned tokens. The pruning mask is then updated by a scatter operation. This keeps the same cascaded pruning rule as the naive implementation, but replaces per-token and per-sample loops with GPU tensor operations.

Population-level parallelism. CMA-ES evaluates a population of candidate schedules in each iteration. Instead of evaluating these candidates independently, we stack the population into a single tensor and evaluate all candidates in parallel. Concretely, the pruning masks, pruning counts, and losses are extended with an additional population dimension. This allows a single batched pass over the calibration data to return the fidelity losses for all candidate schedules. The theoretical cost term $\mathcal{G}(\mathcal{R})$ is also computed in a vectorized manner for the whole population, after which the penalty term is added to obtain the final fitness values.

Summary. Overall, the optimized implementation accelerates objective evaluation through four complementary techniques: loading calibration scores once into GPU batches, grouping samples by length to reduce padding overhead, caching suffix sums to avoid repeated loss accumulation across later layers, and evaluating the entire CMA-ES population in parallel. These optimizations do not change the objective being optimized. They only change how the same objective is computed. This makes the offline schedule search practical even when using a non-gradient optimizer over many candidate pruning schedules.

J Surrogate Objective Loss Sanity Validation

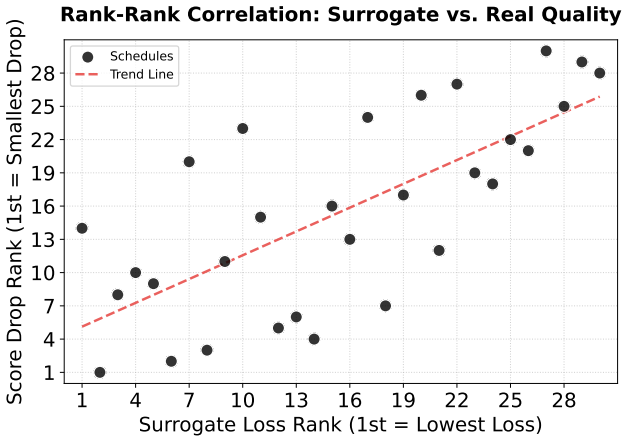

Figure 12: Surrogate-loss ranking vs. LongBench score drop. Each point is a sampled pruning schedule.

The surrogate objective provides a useful ranking signal for selecting pruning schedules. To validate this, we sample 30 schedules with different pruning patterns, compute the surrogate loss for each schedule, and evaluate the corresponding LongBench score drop relative to the full model. We sort the schedules by surrogate loss and analyze the relationship between surrogate-loss rank and score degradation. The results in Fig.12 show that schedules with lower surrogate loss generally preserve quality better, whereas schedules with higher surrogate loss are more likely to cause large performance drops. The Spearman rank correlation between surrogate-loss ranking and LongBench score drop reaches 0.72, indicating that the surrogate objective captures meaningful differences among schedules and can guide the offline search toward quality-preserving pruning strategies.

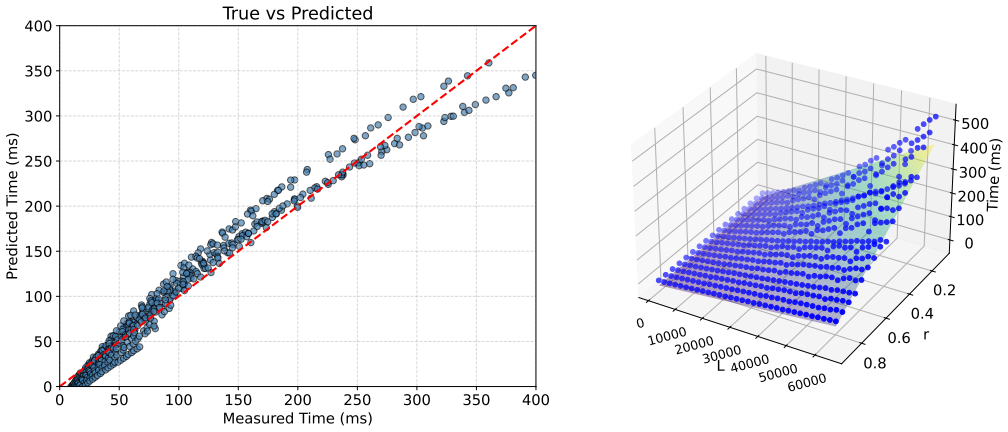

Figure 13: Latency model fitting results for sparse path

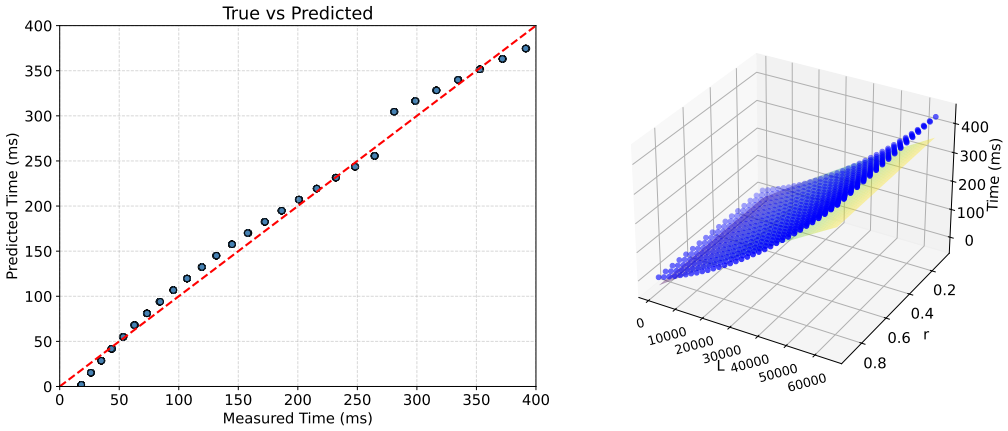

Figure 14: Latency model fitting results for dense path

K Extended Results on Latency Model Fitting

This section provides additional fitting results for the latency models used in the online execution path selection mechanism. Figures 13 and 14 show the fitting results for two representative strategies. The fitted models achieve mean relative errors of 5.16% and 0.63% for strategy 1 and strategy 2, respectively. More importantly, the resulting path-selection accuracy reaches 93.57%, indicating that the model can reliably predict the faster execution path in most cases. These results support the effectiveness of the latency model for online sparse/dense path selection.

L Extended Analysis of Output Consistency

In this section, we provide a sufficient-condition analysis showing when progressive token pruning preserves the top-1 next-token prediction under several simplifying assumptions.

L.1 Notations and Assumptions

Let $f(\cdot)$ denote the Transformer model with L layers. The hidden states at layer l are denoted as $H^{(l)} \in \mathbb{R}^{N\times d}$, where N is the sequence length and d is the hidden dimension. The final logits are $Z = W_{\text{voc}} H^{(L)} \in \mathbb{R}^{|\mathcal{V}|}$, where W_{voc} is the language modeling head.

Assumption 1 (Lipschitz Continuity of Transformer Layers). We assume that each Transformer layer f_l is Lipschitz continuous with respect to the L_∞ norm with a constant $\mathcal{K}_l > 0$. For any hidden states H and $\tilde{H}$:

$$\|f_l(H) - f_l(\tilde{H})\|_\infty \leq \mathcal{K}_l \|H - \tilde{H}\|_\infty$$

Assumption 2 (Bounded Token-Dropping Perturbation) When tokens $\mathcal{P}_l = \mathcal{S}_{l-1} \setminus \mathcal{S}_l$ are dropped, they are excluded from the following computation in layer l. Assume that their omission introduces only a bounded local perturbation to the attention outputs of the surviving tokens, i.e., the local error injected by dropping $\mathcal{P}_l$ is bounded by δ_l:

$$\|f_l(H_{\mathcal{S}_l}^{(l-1)}; K_{\text{all}}, V_{\text{all}}) - f_l(H_{\mathcal{S}_l}^{(l-1)}; K_{\text{surv}}, V_{\text{surv}})\|_\infty \leq \delta_l$$

where $K_{\text{all}}, V_{\text{all}}$ use all tokens in $\mathcal{S}_{l-1}$, and $K_{\text{surv}}, V_{\text{surv}}$ use only the surviving tokens $\mathcal{S}_l$.

L.2 Analysis

(1) In layer l, let $H_{\mathcal{S}_l}^{(l)}$ and $\tilde{H}_{\mathcal{S}_l}^{(l)}$ be the unpruned and pruned hidden state matrices of the surviving tokens at layer l. The cumulative matrix error $\Delta_l = \|H_{\mathcal{S}_l}^{(l)} - \tilde{H}_{\mathcal{S}_l}^{(l)}\|_\infty$ is bounded by: $\Delta_L \leq \sum_{l=1}^{L} \delta_l \prod_{k=l+1}^{L} \mathcal{K}_k$.

To prove it, we introduce a half-hidden states defined as: $H_{\text{half}}^{(l)} = f_l(H_{\mathcal{S}_l}^{(l-1)}; K_{\text{surv}}, V_{\text{surv}})$. Thus

$$\begin{aligned}
\Delta_l &= \|H_{\mathcal{S}_l}^{(l)} - \tilde{H}_{\mathcal{S}_l}^{(l)}\|_\infty \\
&= \|H_{\mathcal{S}_l}^{(l)} - H_{\text{half}}^{(l)} + H_{\text{half}}^{(l)} - \tilde{H}_{\mathcal{S}_l}^{(l)}\|_\infty \\
&\leq \|H_{\mathcal{S}_l}^{(l)} - H_{\text{half}}^{(l)}\|_\infty + \|H_{\text{half}}^{(l)} - \tilde{H}_{\mathcal{S}_l}^{(l)}\|_\infty \\
&\leq (\delta_l) + (\mathcal{K}_l \Delta_{l-1})
\end{aligned}$$

Therefore, unrolling the recurrence from $l = 1$ to L yields the bound Δ_l.

(2) Next, we consider the output distribution of the last token in the prefill phase by the LLM head. Let $y^* = \arg\max_y Z_y$ be the output token of the unpruned model, and $\gamma = Z_{y^*} - \max_{y \neq y^*} Z_y$ be the logit margin. The pruned model via progressive token dropping is guaranteed to output the exact same token y^* if:

$$\Delta_L \leq \sum_{l=1}^{L} \delta_l \prod_{k=l+1}^{L} \mathcal{K}_k < \frac{\gamma}{2\|W_{\text{voc}}\|_\infty}$$

Proof: The final logits are solely determined by the final token's hidden state: $Z = W_{\text{voc}} h^{(L)}$ and $\tilde{Z} = W_{\text{voc}} \tilde{h}^{(L)}$. The maximum logit perturbation is $\|\tilde{Z} - Z\|_\infty \leq \|W_{\text{voc}}\|_\infty \Delta_L$. Thus

$$\|\tilde{Z} - Z\|_\infty \leq \|W_{\text{voc}}\|_\infty \|h^{(L)} - \tilde{h}^{(L)}\|_\infty \leq \|W_{\text{voc}}\|_\infty \Delta_L$$

If $\Delta_L < \frac{\gamma}{2\|W_{\text{voc}}\|_\infty}$, then $\|\tilde{Z} - Z\|_\infty < \frac{\gamma}{2}$. Therefore, this perturbation is strictly less than $\frac{\gamma}{2}$, the original top-1 logit Z_{y^*} cannot decrease by more than $\frac{\gamma}{2}$, and no other logit can increase by more than $\frac{\gamma}{2}$. Consequently, under this sufficient condition, the pruned model preserves the same top-1 next-token prediction as the unpruned model.

In conclusion, this analysis provides intuition for why the offline optimizer favors pruning schedules that avoid removing high-importance tokens, thereby helping preserve model quality.

M Overhead Analysis

M.1 Memory Overhead

The GPU memory overhead in FastPrefill primarily stems from the intermediate recomputed attention score matrices at each layer. This overhead can be formulated as $O(ws \cdot h_{kv} \cdot n)$, where ws is the query

window size, h_{kv} is the number of KV heads, and n is the prompt length. This memory footprint is relatively small. For instance, in Llama3.1-8B with $h_{kv} = 8$, assuming $ws = 64$ and $n = 65536$, the memory overhead is approximately 64 MB (in FP16/BF16 format). This occupies only $\sim 0.4\%$ of the model's 16 GB parameter memory usage. Furthermore, because these intermediate matrices are not permanently stored, the memory is dynamically allocated once and efficiently reused across layers by PyTorch's memory allocator.

M.2 Computational Overhead

Similarly, the computational overhead arises from recomputing the attention score matrices at each layer. The time complexity for this step can be formulated as $O(ws \cdot n \cdot d)$, where d is the hidden dimension size of the model. Note that this computation fundamentally consists of dense matrix multiplications (i.e., $QK^\top$), which are highly parallelizable and executed efficiently by PyTorch's backend on GPUs. Compared to the overall computational cost significantly reduced by FastPrefill, this additional calculation overhead is negligible.

N Limitations

This work focuses on local single-GPU deployment, where long-prompt prefill is often constrained by memory capacity and computational capability. This setting is important for practical local inference, but it does not cover all serving scenarios. In particular, multi-GPU serving introduces additional complexity because both token-importance score computation and sparse attention execution need to be parallelized across devices. Our current implementation does not support multi-GPU scenarios efficiently.

Our method uses an offline calibration set to estimate token importance and determine the layer-wise pruning schedule. This calibration cost is incurred before deployment and can be amortized over later inference requests, but it still introduces an additional preparation step when adapting the method to a new model or workload. In our setting, this cost is acceptable because the resulting schedule can be reused across many prompts, but further reducing the calibration effort would make the method easier to deploy across different models and datasets.

Finally, the quality of the pruning schedule depends on how well the calibration data reflects future inputs. If the inference prompts come from domains that differ significantly from the calibration set, the selected pruning ratios may become less optimal, especially under aggressive speedup targets. In local deployment, this issue can be mitigated by maintaining a public calibration set and periodically adding representative prompts observed during actual usage. The schedule can then be recalibrated periodically, allowing the system to better match the user's workload without changing the online inference procedure.

O Broader Impacts

This work improves the efficiency of long-prompt LLM inference by reducing prefill computation and memory usage. This can have positive impacts by lowering energy consumption, reducing deployment cost, and making long-context models more accessible in local or resource-constrained environments. At the same time, as with other inference acceleration techniques, improved efficiency may also lower the barrier to deploying models for undesirable applications, such as large-scale misinformation generation or other harmful automated uses. We believe such risks should be addressed together with existing model-level safety mechanisms and responsible deployment practices.

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 contributions and scope of the paper are included in the abstract and introduction (Section 1)

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: The limitations are included in the Appendix N.

   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: There is no theoretical result in this paper.

   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 introduce the details of the experiment, which includes the information of hardware and software, models and datasets used, and the details of proposed method in 4.1.

   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: We have attached the data and code used in this paper in the supplementary.

   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 provide the details of the experimental settings in Section 4.1.

   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: We follow previous works by conducting tests on public benchmarks, without including error bars or similar information. For the main results, We use greedy decoding of LLM for stability.

   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: In Section 4.1, we report the details of compute resources, i.e., RTX 4090. We use 8$\times$ RTX4090 to conduct quality experiments and use a single RTX4090 to conduct efficiency experiments.

   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: We have made sure that our paper conforms with the NeurIPS Code of Ethics.

   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: [Yes]

    We discuss the potential positive impacts of proposed method in Section O.

    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: There is no risk of misuse of the proposed method because its purpose is to accelerate the large language model.

    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 have cited the original paper or attached the link to the existing assets used in Section 4.1.

    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: [Yes]

Justification: We have attached the introduction for running code in the code repository.

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 paper 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 paper does not involve.

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: [Yes]

Justification: We describe the usage of LLMs in Section 4.1.

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.