HybridCache: Enhance Prefix Caching of Hybrid LLMs for Efficient Inference

Xin Wang[1,2,*,‡] **Hao Yu**[1,*] **Yi Zhang**[1] **Huiqiang Jiang**[1] **Bo Zheng**[1]
Mi Zhang[2,†] **Dayiheng Liu**[1] **Jianwei Zhang**[1,†]

[1]Alibaba Token Hub, Alibaba Group, [2]Ohio State University

Abstract

The efficiency of Large Language Model (LLM) serving has been significantly improved by prefix caching, which enables reuse of previously computed activations. However, extending prefix caching to hybrid linear–softmax language models remains challenging due to the presence of large recurrent states in linear attention modules. In practice, these states cannot be stored at every token position due to their substantial memory footprint, and are instead saved periodically (e.g., at chunk boundaries). This leads to a fundamental mismatch between token-level KV caching and coarse-grained state caching, resulting in reduced effective cache hit rates and additional recomputation overhead. In this work, we propose `HybridCache`, a system designed to address the limitations of prefix caching in hybrid linear–softmax LLMs. `HybridCache` is built on two key observations: (1) recurrent states exhibit different degrees of state variation across heads, and (2) the distribution of state values presents structured outliers that hinder effective compression. Based on these insights, `HybridCache` introduces two key designs. First, we develop a recurrent-aware state reuse mechanism that selectively caches head states with significant changes while reusing previously cached states for relatively stable heads, enabling fine-grained alignment with KV caching without incurring prohibitive memory cost. Second, we propose a dual-scale state quantization method that applies scaling along both row and column dimensions to effectively compress states with structured outliers. We evaluate `HybridCache` on hybrid linear–softmax LLMs under realistic serving workloads. Experimental results show that `HybridCache` improves effective prefix cache utilization, reduces recomputation overhead, and achieves substantial throughput gains while maintaining comparable model accuracy.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities across a wide range of tasks. To support high-throughput serving, prefix caching has become a critical system optimization that reuses previously computed activations to significantly reduce redundant computation. While prefix caching is well understood and implemented [5, 17] for Transformer-based models with softmax attention, its extension to hybrid linear–softmax architectures [12, 11, 16, 8, 7] introduces new challenges. Specifically, both key–value (KV) caches for softmax attention and recurrent states for linear attention need to be cached for prefix matching in hybrid linear–softmax LLMs. However, unlike KV caches that are stored at fine granularity, linear attention states are substantially larger, making it prohibitively expensive to cache them densely [10]. For example, caching 1024 tokens'

*Equal contribution.
†Corresponding author.
‡Work is done during internship at Alibaba Token Hub, Alibaba Group.

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

recurrent states for Qwen-3.5 35BA3B consumes 460 GB, which is about 3000 times larger than the corresponding KV cache.

To avoid the out-of-memory (OOM) issue of caching recurrent states, existing LLM serving systems [5, 17, 9] such as vLLM and SGLang typically retain recurrent states only at coarse-grained checkpoints (e.g., chunk boundaries). Despite its practicality, this approach introduces a fundamental mismatch in cache granularity: KV caches are maintained at fine granularity, while recurrent states are only available at sparse checkpoints. As a result, a prefix match may hit the KV cache at a token position where the corresponding recurrent state is unavailable, forcing the system to recompute the missing state from the nearest checkpoint. This additional recomputation reduces the effective cache hit ratio and limits the latency benefits of prefix caching for serving hybrid models.

To address this limitation, we propose `HybridCache`, a new caching framework tailored for hybrid linear–softmax LLMs. `HybridCache` resolves the mismatch between KV and state caching while maintaining memory efficiency through two key designs. ❶ **Recurrent-Aware State Reuse:** `HybridCache` selectively caches head states with significant deviations while reusing previously cached states for relatively stable heads. This enables fine-grained cache alignment between KV and state caching without incurring the cost of dense storage. ❷ **Dual-Scale State Quantization:** To reduce the storage cost of recurrent states, `HybridCache` incorporates a dual-scale quantization approach that applies successive scaling along row and column dimensions to handle structured outliers.

We evaluate `HybridCache` on five representative hybrid linear–softmax LLMs from three different LLM families under four realistic serving workloads covering agentic, long-context, and reasoning tasks. Our experiments show that `HybridCache` improves cache utilization, reduces recomputation overhead, and achieves lower latency and higher throughput while maintaining comparable task accuracy. We highlight four main findings:

- `HybridCache` consistently improves effective prefix cache utilization across different hybrid linear–softmax models and workloads by mitigating KV–state granularity mismatch.
- `HybridCache` maintains comparable accuracy across most evaluated workloads under recurrent-aware state reuse and dual-scale state quantization, demonstrating that these optimizations introduce limited quality degradation.
- `HybridCache` reduces serving latency while improving throughput, with more pronounced benefits on larger models and longer-prefix workloads.
- `HybridCache` consistently enhances end-to-end serving efficiency under various practical serving settings, including different batch sizes and chunk prefill sizes.

2 Related Work

Linear–Softmax Language Models. Recent efforts on efficient sequence modeling aim to overcome the quadratic complexity of softmax attention by introducing linear-time alternatives. Linear attention methods reformulate attention using kernel feature maps to achieve $\mathcal{O}(N)$ complexity [4]. Subsequent works, including Performer [1], further improve the scalability of attention mechanisms. More recently, state-space models such as Mamba [2] introduce selective state representations that enable efficient recurrent computation while maintaining strong performance. To balance efficiency and modeling capacity, hybrid linear–softmax architectures have emerged, combining softmax attention with linear attention or state-space modules [12, 11, 16, 8, 7]. These models retain the expressiveness of softmax attention while benefiting from the efficiency of recurrent mechanisms, and have been widely adopted in modern LLM systems. However, unlike Transformer models that rely solely on KV caches, hybrid architectures introduce recurrent states that differ substantially in structure and memory footprint. Moreover, recurrent states exhibit varying degrees of temporal variation across heads, making uniform state caching inefficient. These characteristics motivate caching strategies that account for head-dependent state evolution while minimizing storage overhead.

Prefix Caching for LLM Serving. Prefix caching is a key system optimization for LLM serving, enabling reuse of previously computed activations to reduce redundant computation and improve throughput. Modern serving systems such as vLLM [5] leverage KV cache management techniques (e.g., PagedAttention) to efficiently store and reuse intermediate activations across requests. SGLang [17] further employs radix-tree-based caching to support efficient prefix sharing and reuse.

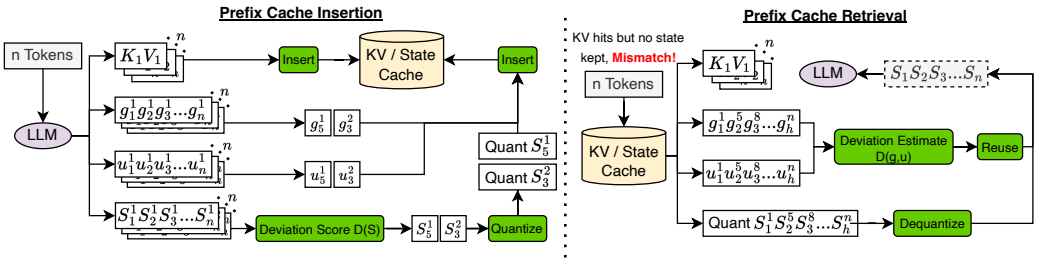


Figure 1: Overview of `HybridCache`. During insertion, `HybridCache` selectively caches high-deviation recurrent head states and compact statistics for subsequent reuse decisions. During retrieval, it estimates the deviation of unavailable states and reuses cached states only when the estimated deviation is sufficiently small. Cached states are compressed with dual-scale FP4 quantization.

These systems organize KV caches in a block-wise manner, allowing flexible memory allocation and sharing across requests, and have become widely adopted for high-throughput LLM serving. However, these approaches are primarily designed for Transformer-based models, where KV tensors can be reused at fine granularity. In hybrid linear–softmax models, recurrent states are substantially larger and cannot be stored densely due to memory constraints. Consequently, practical systems typically retain recurrent states at coarse-grained checkpoints, introducing a mismatch between KV and recurrent-state caching granularity. This mismatch limits effective cache reuse and introduces additional recomputation overhead. Recent work such as Marconi [9] addresses prefix caching for hybrid LLMs through specialized cache admission and eviction policies. By considering potential reuse opportunities and computation savings relative to memory consumption, Marconi improves cache utilization while avoiding excessive recurrent-state storage. However, its primary focus is on selecting which cache entries to retain, rather than exploiting head-dependent state variations to enable finer-grained reuse. In contrast, `HybridCache` selectively caches head states with significant deviations and reuses previously cached states for relatively stable heads, mitigating the KV–state granularity mismatch without requiring dense state storage. Furthermore, existing cache compression techniques primarily focus on KV tensors, while recurrent states exhibit different numerical characteristics due to their accumulated outer-product updates. In particular, structured outliers along both row and column dimensions make conventional single-scale quantization less effective. To address this limitation, `HybridCache` introduces dual-scale state quantization, complementing recurrent-aware state reuse to further improve memory efficiency and prefix cache utilization.

3 HybridCache

`HybridCache` improves prefix caching for hybrid linear–softmax language models by addressing the mismatch between fine-grained KV cache entries and coarse-grained recurrent-state checkpoints. It combines two complementary designs. *Recurrent-aware state reuse* selectively materializes states that change substantially and estimates whether previously cached states can be reused at an unmatched position. *Dual-scale state quantization* reduces the memory cost of cached recurrent states while accounting for outliers distributed across both matrix dimensions. Figure 1 illustrates the workflow.

3.1 Preliminaries

Hybrid linear–softmax models combine softmax-attention KV caches with recurrent states maintained by linear-attention or state-space layers. Consider a Gated DeltaNet head h with matrix-valued state $S_t^{(h)} \in \mathbb{R}^{M\times N}$, row-vector key $k_t^{(h)} \in \mathbb{R}^{1\times M}$, and row-vector value $v_t^{(h)} \in \mathbb{R}^{1\times N}$. Its recurrent update can be written as

$$S_t^{(h)} = \alpha_t^{(h)}\left(I - \beta_t^{(h)} k_t^{(h)\top} k_t^{(h)}\right) S_{t-1}^{(h)} + \beta_t^{(h)} k_t^{(h)\top} v_t^{(h)}, \tag{1}$$

where $\alpha_t^{(h)} = \exp(g_t^{(h)}) \in (0,1]$ controls decay and $\beta_t^{(h)} \in (0,1)$ controls the delta update. Rearranging Eq. (1) yields

$$S_t^{(h)} = \alpha_t^{(h)} S_{t-1}^{(h)} + U_t^{(h)}, \quad U_t^{(h)} = \beta_t^{(h)} k_t^{(h)\top}\left(v_t^{(h)} - \alpha_t^{(h)} k_t^{(h)} S_{t-1}^{(h)}\right). \tag{2}$$

Thus, $U_t^{(h)}$ includes both the new write and the delta correction; Eq. (2) is an exact rearrangement rather than an approximation that discards the correction. A gated outer-product recurrence, such as $S_t^{(h)} = \alpha_t^{(h)} S_{t-1}^{(h)} + k_t^{(h)\top} v_t^{(h)}$, fits the same form with a different $U_t^{(h)}$. We use the unified notation below whenever the corresponding decay and write terms are available.

During prefill, recurrent states are computed over chunks, whereas decoding advances them token by token. Unlike softmax-attention KV entries, a recurrent state summarizes the entire preceding prefix and can be substantially more expensive to retain at every token position. Practical hybrid prefix caches therefore retain full recurrent states at selected checkpoints, potentially leaving a KV hit without a matching recurrent state.

3.2 Recurrent-Aware State Reuse

Motivation. A KV cache hit at token u cannot be fully exploited when the recurrent state S_u is unavailable: conventional serving must recover the state by replaying tokens from a preceding checkpoint. However, recurrent heads do not evolve uniformly. Some change substantially over a short interval, while others remain sufficiently stable that a nearby cached state is a useful approximation. Importantly, state stability depends on both historical decay and new writes; the decay gate alone does not determine whether a head can be safely reused. `HybridCache` therefore uses state deviation to guide selective storage and reuse.

Design. `HybridCache` combines selective state caching with deviation-guided retrieval. During insertion, it prioritizes head states that change substantially; during retrieval, it uses lightweight recurrent statistics to decide whether previously cached states can be reused. The following components implement these steps efficiently.

(1) Deviation-Aware State Selection. For head h, a cached position e, and a later position u, define

$$D_h(e \to u) \triangleq \left\| S_u^{(h)} - S_e^{(h)} \right\|_F . \tag{3}$$

A larger D_h means that substituting $S_e^{(h)}$ for $S_u^{(h)}$ incurs a larger state-space error, irrespective of whether that change originates from decay or new writes. During cache insertion, when the states needed to measure Eq. (3) are available, `HybridCache` ranks head states by deviation relative to the corresponding retained reference state and preferentially stores the top-K high-deviation states. Lower-deviation heads can instead refer to previously retained states. This enables more fine-grained state coverage without densely storing every head state at every token position. The selected states and the statistics needed for future reuse decisions are associated with the prefix cache and compressed using Section 3.3.

(2) Deviation Estimation. At retrieval time, the target state $S_u^{(h)}$ is not available, so Eq. (3) cannot be evaluated directly. Unrolling Eq. (2) from the reference position e gives

$$S_u^{(h)} = R_h(e \to u) S_e^{(h)} + \sum_{i=e+1}^{u} R_h(i \to u) U_i^{(h)}, \; R_h(a \to b) \triangleq \exp\left(\sum_{t=a+1}^{b} g_t^{(h)} \right). \tag{4}$$

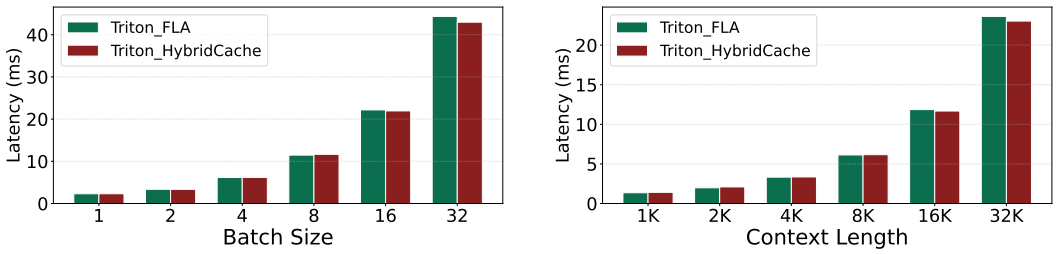
Because $0 < R_h(a \to b) \le 1$, the triangle inequality yields

$$D_h(e \to u) \le \big(1 - R_h(e \to u)\big) \left\| S_e^{(h)} \right\|_F + \sum_{i=e+1}^{u} R_h(i \to u) \left\| U_i^{(h)} \right\|_F . \tag{5}$$

This bound separates the contribution of decaying the reference state from the contribution of subsequent writes. The gate values and write magnitudes can be collected during prefill, allowing a reuse decision without first reconstructing the missing full state.

(3) Chunk-Level Estimation and Cache Retrieval. To limit metadata cost, `HybridCache` uses chunk-level write statistics rather than storing every full update matrix. Let $\mathcal{C}(e, u)$ denote the chunks intersecting $(e, u]$, and let $W_c^{(h)}$ be the stored write statistic for chunk c. We estimate Eq. (5) as

$$\widehat{D}_h(e \to u) = \big(1 - R_h(e \to u)\big) \left\| S_e^{(h)} \right\|_F + \sum_{c \in \mathcal{C}(e,u)} \omega_h(c, u) W_c^{(h)}, \tag{6}$$



(a) Various batch size

(b) Various Context Length

Figure 2: Prefill-kernel latency of the original Triton FLA implementation and the `HybridCache` variant under different batch sizes and sequence lengths.

where $\omega_h(c, u)$ approximates the decay applied to the writes in chunk c before reaching u. In particular, an aggregated chunk-update norm need not upper-bound the sum of token-wise update norms because writes may cancel. We therefore treat Eq. (6) as an *estimator*, not an unconditional theoretical upper bound; its precise aggregation and decay weighting follow the implementation.

On a prefix hit, `HybridCache` restores a head from its cached state when one is available at the matched position. Otherwise, it estimates the deviation between the missing state and the head's available reference state. The reference is reused only when $\widehat{D}_h(e \to u) \leq \sigma$, where σ is a configurable threshold. If the available cached states cannot satisfy this criterion for the required recurrent heads, `HybridCache` falls back to recurrent-state recomputation rather than accepting an unchecked approximation. This conservative retrieval policy balances effective prefix reuse against approximation error.

(4) Fused State-Statistics Collection and Cache Integration. We augment the Triton-based linear-attention prefill kernel from Flash Linear Attention (FLA) [13] to collect head-state norms and write statistics during the existing chunkwise computation. Fusing the reductions with prefill avoids separately materializing the full update tensors for caching. Figure 2 compares the original and modified kernels across batch sizes and prefill lengths on an NVIDIA A100 GPU and shows limited additional latency. The cached statistics are used only to guide subsequent prefix-cache retrieval; they do not alter the model's recurrent update rule.

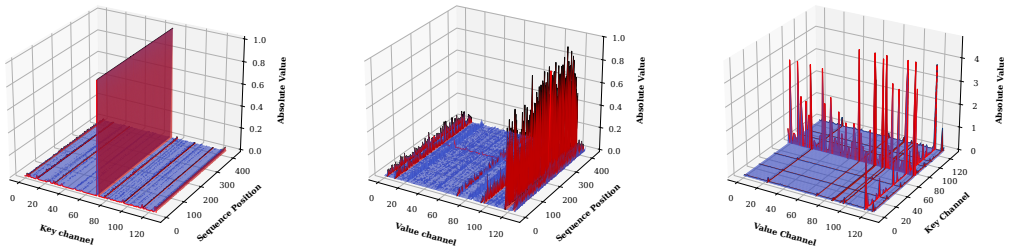
3.3 Dual-Scale State Quantization

Motivation. Selective state caching still incurs substantial memory cost, particularly for large recurrent-state matrices. Although 4-bit quantization can reduce that cost, conventional single-scale schemes are sensitive to structured outliers. In a recurrent update containing an outer product $k_t^\top v_t$, atypically large key and value channels affect entire rows and columns of the state matrix. Figure 3 visualizes this behavior for Qwen3.5-35B-A3B. A single global scale must accommodate the largest values, while scaling along only one dimension cannot independently adapt to outliers along the other. We therefore apply two successive, orthogonal scaling steps before FP4 encoding.

Design. `HybridCache` applies two successive scaling steps to accommodate outliers along both dimensions of recurrent states. We first describe the quantization scheme and its storage cost, then explain how fused kernels integrate it into prefix caching.

(1) Dual-Scale State Quantization. Consider a recurrent-state tensor $S \in \mathbb{R}^{LH \times M \times N}$, where L and H are the numbers of recurrent layers and heads. We quantize each $M \times N$ head-state matrix independently using groups of size G along each axis. For each head state, the first scale s_M is computed over groups of G elements along M; it has shape $(M/G) \times N$. After normalizing by s_M, the second scale s_N is computed over groups of G elements along N; it has shape $M \times (N/G)$. With broadcasting over the grouped axes, the quantized payload and reconstructed state take the form

$$Q = \mathcal{Q}_{\mathrm{FP4}}\left(\frac{S}{s_M \odot s_N}\right), \qquad \widehat{S} = \mathcal{D}_{\mathrm{FP4}}(Q) \odot s_M \odot s_N, \tag{7}$$



(a) k slice: channel-wise outliers (b) v slice: channel-wise outliers (c) state: row- and column-wise outliers

Figure 3: Channel-wise outliers in k and v produce structured outliers along both dimensions of the recurrent-state matrix through outer-product updates.

where $\odot$ denotes elementwise multiplication with the appropriate broadcasting and the two scale tensors are stored in FP16. The scale normalization incorporates the finite range of the adopted FP4 format (maximum magnitude $V_{\max}^{\mathrm{FP4}} = 6$).

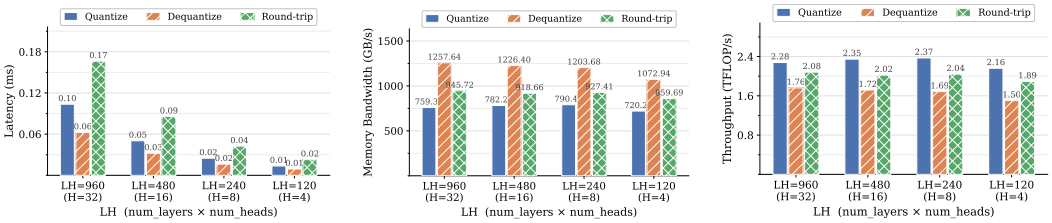
The FP4 payload requires 4 bits per element. For the two stated scale layouts, the FP16 scale metadata costs $16/G$ bits per element for each axis, giving

$$b_{\mathrm{eff}} = 4 + \frac{32}{G} \quad \text{bits per state element.} \tag{8}$$

At $G = 16$, this is 6 bits per element, corresponding to $5.3\times$ compression relative to an FP32 state (or $2.7\times$ relative to BF16), excluding any additional allocator or alignment overhead.

(2) Fused Kernels and Cache Integration. We implement quantization and dequantization as fused Triton kernels to avoid materializing intermediate normalized tensors. Each quantization program processes a $G \times G$ tile, computes the two scales in sequence, maps normalized values to the FP4 range, and packs two FP4 values into each byte. Dequantization reverses the process and applies both stored scales to reconstruct the state. All scale operations are performed in registers where possible.

The compressed tuple $(Q_{\mathrm{FP4}}, s_M, s_N)$ is integrated into the radix-cache lifecycle. On insertion, the cache retains the compressed state and its scale metadata; on a cache hit, the state is dequantized into the destination recurrent-state slot before the subsequent forward pass. Cache ownership and eviction accounting continue to use the existing pool-slot and radix-tree bookkeeping, so compression does not require a separate cache-management policy.



(a) Kernel latency (b) Effective memory bandwidth (c) Effective compute throughput

Figure 4: Microbenchmark of the fused FP4 cache kernels as the number of layer–head states (LH) changes. Round-trip includes quantization and dequantization.

Figure 4 reports kernel performance for LH between 120 and 960. The full quantization–dequantization round trip takes at most about 0.17 ms in these configurations, with lower latency at smaller LH. The measurements indicate that dual-scale compression can be implemented with modest per-operation overhead; its end-to-end benefit is evaluated separately through cache utilization and serving latency.

Table 1: Performance including accuracy (%), cache hit ratio (%), and TTFT (s) on one agentic benchmark DeepPlanning, one long-context benchmark RULER-98K, and two reasoning benchmarks ZebraLogic and LongProc with three representative linear-softmax language models, including Qwen-3.5 35BA3B , Kimi-Linear, and Nemotron-3-Nano. The best performance is marked in bold.

Model	Method	DeepPlanning			RULER-98K			ZebraLogic			LongProc		
		Acc (↑)	Hit Ratio (↑)	TTFT (↓)	Acc (↑)	Hit Ratio (↑)	TTFT (↓)	Acc (↑)	Hit Ratio (↑)	TTFT (↓)	Acc (↑)	Hit Ratio (↑)	TTFT (↓)
Qwen	SGLang	**40.00**	88.73	0.67	**97.63**	33.86	0.93	**88.14**	52.90	0.13	80.92	58.25	0.17
	HybridCache	39.87	**93.13**	**0.31**	97.56	**67.61**	**0.66**	87.28	**65.73**	**0.09**	**81.43**	**67.62**	**0.14**
Kimi	SGLang	**24.17**	85.92	0.88	**95.78**	47.63	3.29	42.83	63.97	0.12	27.57	79.97	0.20
	HybridCache	19.67	**90.34**	**0.63**	95.65	**95.13**	**0.63**	**43.14**	**72.18**	**0.08**	**27.79**	**88.08**	**0.17**
Nemo	SGLang	**32.50**	87.33	0.78	**94.44**	66.95	0.88	**43.41**	71.78	0.13	**44.53**	77.11	1.73
	HybridCache	30.88	**92.81**	**0.45**	94.30	**94.62**	**0.32**	42.78	**84.76**	**0.09**	43.64	**96.73**	**1.23**

Table 2: Performance of HybridCache and original SGLang Baseline on RULER-98K with Qwen-3.5 model family [12] convering various scales, including 35BA3B, 122BA10B, and 397BA17B. The best performance is marked in bold.

Method	Qwen-35BA3B		Qwen-122BA10B		Qwen-397BA17B	
	Acc (↑)	TTFT (↓)	Acc (↑)	TTFT (↓)	Acc (↑)	TTFT (↓)
SGLang	**97.63**	0.93	97.31	1.63	97.69	2.47
HybridCache	97.56	**0.66**	**97.31**	**1.03**	**98.08**	**1.41**

Table 3: Performance of HybridCache on Qwen-3.5 35BA3B with recurrent-aware state reuse only (denoted as HybridCache (R)), HybridCache with dual-scale state quantization only (denoted as HybridCache (D)), full HybridCache, and the original SGLang. The best performance is marked in bold.

Metric	HybridCache (R)	HybridCache (D)	HybridCache	SGLang
Acc (↑)	98.43	97.95	**97.56**	97.63
TTFT (↓)	0.74	0.52	**0.66**	0.93

4 Experiments and Analysis

Baselines. We compare HybridCache against three existing caching strategies for linear-softmax language models: (1) the default strategy used in SGLang [17], (2) the default strategy used in vLLM [5], and (3) Marconi [9], a recent prefix caching approach for hybrid LLMs.

Models and Datasets. To demonstrate the generalizability of HybridCache, we evaluate its performance on five hybrid LLMs from four different families at different scales (Qwen-3.5 35BA3B, 122BA10B, 397BA17B [12], Kimi-Linear [16], and Nemotron-3-Nano [8]) and four representative benchmarks, including one agentic benchmark DeepPlanning [15], one long-context benchmark RULER-98K [3], and two reasoning benchmarks ZebraLogic [6] and LongProc [14].

Experimental Details. To ensure a fair comparison, we implemented and evaluated HybridCache and all three baseline methods in the same LLM serving framework SGLang [17]. Unless otherwise specified, all evaluations are conducted with a maximum batch size of 16, max mamba state size = 256, Tensor Parallel Size = 8, with triton Linear attention backend on 8 GPUs. The chunk prefill size is provided in Appendix E. We measured the average cache hit rate, time-to-first-token (TTFT), per-request latency and throughput along with task accuracy.

4.1 Overall Comparison

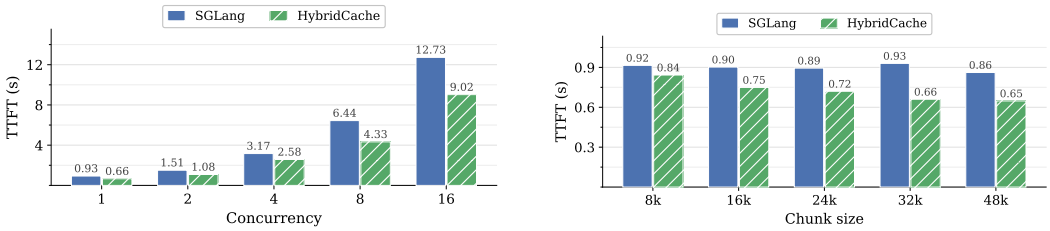
We compare HybridCache with the default prefix caching strategy in SGLang across different models and workloads. Table 1 reports accuracy, cache hit ratio, and TTFT. Overall, HybridCache consistently improves cache hit ratio and reduces TTFT while maintaining comparable task accuracy.

Performance on Different Models. Across Qwen-3.5 35BA3B, Kimi-Linear, and Nemotron-3-Nano, HybridCache consistently improves effective cache utilization. On RULER-98K, HybridCache increases the cache hit ratio from 33.86% to 67.61% on Qwen, from 47.63% to 95.13% on Kimi, and from 66.95% to 94.62% on Nemotron. These improvements translate into clear TTFT reduction, e.g., from 0.93 to 0.66 on Qwen, from 3.29 to 0.63 on Kimi, and from 0.88 to 0.32 on Nemotron. Similar gains are also observed on ZebraLogic and LongProc, showing that HybridCache generalizes across different hybrid linear–softmax architectures.

Performance Under Different Workloads. HybridCache achieves consistent improvements across long-context and reasoning workloads. On RULER-98K, where long shared prefixes make state recomputation particularly costly, HybridCache substantially improves cache hit ratio and reduces TTFT for all evaluated models. On ZebraLogic and LongProc, HybridCache also improves cache

Table 4: Accuracy (%) on RULER with Qwen-3.5 35BA3B under different prefill context lengths. The best performance is marked in bold.

METHOD	Acc (↑) Across Various Prefill Context Lengths										
	65K	98K	131K	196K	262K	393K	524K	655K	786K	918K	Avg.
SGLang	97.95	**97.63**	96.92	**97.31**	97.69	**97.27**	**96.54**	**96.33**	**94.33**	**93.78**	**96.58**
`HybridCache`	**98.08**	97.56	**97.69**	96.83	**97.88**	96.88	96.06	96.00	94.12	92.79	96.39



(a) Maximum batch size

(b) Chunk prefill size

Figure 5: TTFT comparison under different maximum batch sizes and chunk prefill sizes.

hit ratio by 8–20 percentage points in most cases and reduces TTFT accordingly. Accuracy remains comparable on most workloads, although some configurations exhibit larger deviations from SGLang.

Performance on Larger-Scale LLMs. Table 2 evaluates `HybridCache` on the Qwen-3.5 model family at different scales. `HybridCache` reduces TTFT from 0.93 to 0.66 on Qwen-35BA3B, from 1.63 to 1.03 on Qwen-122BA10B, and from 2.47 to 1.41 on Qwen-397BA17B, while maintaining comparable accuracy. The larger gain on Qwen-397BA17B shows that `HybridCache` becomes increasingly beneficial as wasted prefill compute becomes more expensive in larger hybrid models.

4.2 Ablation Study

Modular Sensitivity Study. We first study the contribution of each component in `HybridCache`. As shown in Table 3, using recurrent-aware state reuse only, denoted as `HybridCache` (R), achieves the highest accuracy of 98.43 with TTFT of 0.74. Using dual-scale state quantization only, denoted as `HybridCache` (D), further reduces TTFT to 0.52 while maintaining accuracy of 97.95. Combining both designs, `HybridCache` achieves 97.56 accuracy and 0.66 TTFT. These results show that both designs improve TTFT individually, while their combination does not yield additive latency gains in this setting. The full `HybridCache` nevertheless reduces TTFT relative to SGLang while maintaining comparable accuracy.

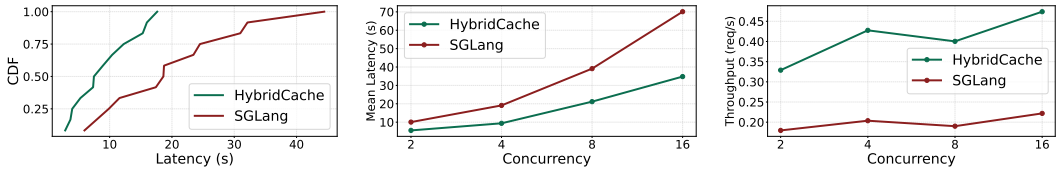
Performance Under Different Prefill Context Lengths. Table 4 evaluates accuracy on RULER under different prefill context lengths. `HybridCache` maintains comparable accuracy to SGLang across all context lengths. Notably, `HybridCache` achieves higher accuracy than SGLang at 131K and 262K context lengths. These results indicate that recurrent-aware state reuse does not introduce noticeable accuracy degradation even for long-prefix workloads.

Performance Under Different Concurrency. Figure 5(a) reports TTFT under different maximum batch sizes. `HybridCache` consistently reduces TTFT compared with SGLang across all evaluated maximum batch sizes. The improvement becomes more pronounced at larger batch sizes, where the cost of recurrent-state recomputation is more substantial.

Performance Under Different Chunk Prefill Size. Figure 5(b) shows TTFT under different chunk prefill sizes. `HybridCache` consistently outperforms SGLang across all chunk sizes. The gain is more visible for larger chunk sizes, where each cache mismatch may trigger more recurrent-state recomputation. This confirms that `HybridCache` effectively mitigates the KV–state granularity mismatch under different prefill configurations.

4.3 Analysis Under Other System Metrics

E2E Latency Analysis. Figure 6(a) and Figure 6(b) report the end-to-end latency of `HybridCache` and SGLang on RULER-98K under different concurrency levels. `HybridCache` consistently achieves



(a) Latency Distribution (b) Average Request Latency (s) (c) Average Throughput (req/s)

Figure 6: Performance comparison with other system metrics, including tail latency (s), average request latency (s), and average throughput (req/s) on Qwen-3.5 35BA3B in RULER-98K benchmark under varying concurrency levels.

Table 5: Performance on RULER-98K with Qwen-3.5 35BA3B using different caching optimization strategies. The best performance is marked in bold.

METHOD	Cache Reuse			Cache Quantization			Cache Eviction		
	Direct	Zero	Recurrent	Single Int4	Single FP4	Dual FP4	vLLM	Marconi	`HybridCache`
Acc (↑)	94.26	82.73	**97.56**	95.38	96.89	**97.56**	**97.61**	96.98	97.56
TTFT (↓)	**0.62**	0.64	0.66	0.66	0.66	**0.66**	0.98	0.92	**0.66**

lower latency. In particular, the latency CDF shifts significantly to the left, showing that `HybridCache` reduces both average and tail latency. As concurrency increases, the advantage becomes more evident. This demonstrates that reducing recurrent-state recomputation is especially beneficial under high-concurrency serving.

Throughput Analysis. Figure 6(c) compares average throughput under different concurrency levels. `HybridCache` consistently outperforms SGLang across all settings. These results show that `HybridCache` not only reduces latency but also improves overall serving capacity by converting more prefix matches into effective cache hits.

4.4 Comparison with Existing Cache Optimization Strategies

Cache Reuse. We first compare different state reuse strategies in Table 5. Directly reusing the nearest cached state achieves low TTFT but reduces accuracy due to inaccurate state reconstruction. Zeroing the missing state further degrades accuracy to 82.73. In contrast, our recurrent-aware reuse achieves much higher accuracy with comparable TTFT. This shows that selectively reusing long-range heads is more effective than applying a uniform reuse policy to all recurrent states.

Cache Quantization. We then compare different cache quantization strategies. Single-scale INT4 and FP4 quantization achieve similar TTFT but with lower accuracy. This suggests that dual-scale quantization better accommodates the structured outliers in recurrent states than the evaluated single-scale alternatives.

Cache Eviction. Finally, we compare `HybridCache` with existing cache eviction strategies for hybrid models, including vLLM and Marconi. While both maintain competitive accuracy, their TTFT remains higher at 0.98 and 0.92, respectively. In contrast, `HybridCache` achieves similar accuracy with a lower TTFT of 0.66, showing that eviction-only optimization cannot fully address the KV–state granularity mismatch in hybrid linear–softmax models.

5 Conclusion

In this work, we present `HybridCache`, a hybrid prefix caching framework for linear–softmax language models. `HybridCache` enables fine-grained recurrent state reuse aligned with token-level KV caching and introduces dual-scale quantization to compress states with structured outliers. Experiments show that `HybridCache` improves cache utilization and serving throughput while preserving model accuracy.

Acknowledgements

This work was supported by Alibaba Research Intern Program.

References

[1] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J. Colwell, and Adrian Weller. Rethinking attention with performers. In *ICLR*. OpenReview.net, 2021.

[2] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *CoRR*, abs/2312.00752, 2023.

[3] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: what's the real context size of your long-context language models? *CoRR*, abs/2404.06654, 2024.

[4] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *ICML*, Proceedings of Machine Learning Research, pages 5156–5165. PMLR, 2020.

[5] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *SOSP*, pages 611–626. ACM, 2023.

[6] Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. Zebralogic: On the scaling limits of llms for logical reasoning. In *ICML*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025.

[7] William Merrill, Yanhong Li, Tyler Romero, Anej Svete, Caia Costello, Pradeep Dasigi, Dirk Groeneveld, David Heineman, Bailey Kuehl, Nathan Lambert, et al. Olmo hybrid: From theory to practice and back. *arXiv preprint arXiv:2604.03444*, 2026.

[8] NVIDIA. Nemotron 3 Nano: Open, efficient mixture-of-experts hybrid Mamba-Transformer model for Agentic reasoning, 2025. URL `https://arxiv.org/abs/2512.20848`. Technical report.

[9] Rui Pan, Zhuang Wang, Zhen Jia, Can Karakus, Luca Zancato, Tri Dao, Yida Wang, and Ravi Netravali. Marconi: Prefix caching for the era of hybrid llms. In *MLSys*. OpenReview.net/mlsys.org, 2025.

[10] Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free. *CoRR*, abs/2505.06708, 2025.

[11] Qwen Team. Qwen3.6-Plus: Towards real world agents, April 2026. URL `https://qwen.ai/blog?id=qwen3.6`.

[12] Qwen Team. Qwen3.5: Accelerating productivity with native multimodal agents, February 2026. URL `https://qwen.ai/blog?id=qwen3.5`.

[13] Songlin Yang and Yu Zhang. Fla: A triton-based library for hardware-efficient implementations of linear attention mechanism, January 2024. URL `https://github.com/fla-org/flash-linear-attention`.

[14] Xi Ye, Fangcong Yin, Yinghui He, Joie Zhang, Howard Yen, Tianyu Gao, Greg Durrett, and Danqi Chen. Longproc: Benchmarking long-context language models on long procedural generation. *CoRR*, abs/2501.05414, 2025.

[15] Yinger Zhang, Shutong Jiang, Renhao Li, Jianhong Tu, Yang Su, Lianghao Deng, Xudong Guo, Chenxu Lv, and Junyang Lin. Deepplanning: Benchmarking long-horizon agentic planning with verifiable constraints. *CoRR*, abs/2601.18137, 2026.

[16] Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, Wentao Li, Enzhe Lu, Weizhou Liu, Yanru Chen, Weixin Xu, Longhui Yu, Yejie Wang, Yu Fan, Longguang Zhong, Enming Yuan, Dehao Zhang, Yizhi Zhang, T. Y. Liu, Haiming Wang, Shengjun Fang, Weiran He, Shaowei Liu, Yiwei Li, Jianlin Su, Jiezhong Qiu, Bo Pang, Junjie Yan, Zhejun Jiang, Weixiao Huang, Bohong Yin, Jiacheng You, Chu Wei, Zhengtao Wang, Chao Hong, Yutian Chen, Guanduo Chen, Yucheng Wang, Huabin Zheng, Feng Wang, Yibo Liu, Mengnan Dong, Zheng Zhang, Siyuan Pan, Wenhao Wu, Yuhao Wu, Longyu Guan, Jiawen Tao, Guohong Fu, Xinran Xu, Yuzhi Wang, Guokun Lai, Yuxin Wu, Xinyu Zhou, Zhilin Yang, and Yulun Du. Kimi linear: An expressive, efficient attention architecture. *CoRR*, abs/2510.26692, 2025.

[17] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In *NeurIPS*, 2024.

A Limitations

`HybridCache` targets hybrid linear–softmax LLMs and may not directly apply to pure Transformers. Its deviation-based state reuse may be less accurate under highly dynamic recurrent updates. Additionally, storing gate values and write statistics introduces extra metadata and system complexity.

B Broader Impact

`HybridCache` improves LLM serving efficiency by reducing redundant computation and cache overhead, potentially lowering deployment costs and energy consumption. However, more efficient serving may also facilitate the large-scale misuse of LLMs. Moreover, approximate state reuse may introduce reconstruction errors, requiring careful evaluation in practical deployments.

C Decoding-Time Cache Reuse

We evaluate cache reuse during decoding on AIME24/25 using Qwen3.5-35A3B with a beam size of 16. We compare the original SGLang baseline with `HybridCache` under different decoding cache reuse gaps, including 256, 1K, 2K, and 4K tokens. As shown in Table 6, `HybridCache` consistently preserves or improves accuracy over the baseline, and the largest gap achieves the best Acc@1.

Table 6: Decoding-time cache reuse results on AIME24/25 using Qwen3.5-35A3B with a beam size of 16.

BENCHMARK	SGLang	`HybridCache` cache reuse gap			
	Baseline	256	1K	2K	4K
AIME24/25 Acc@1	**80.89**	80.00	80.11	80.69	80.81

Table 7: ShareGPT prefix cache reuse results using Qwen3.5-35A3B with a 1K chunk size and $T_p = 2$.

METHOD	TTFT (MS)	CACHE HIT RATE (%)
Baseline	368.28	34.56
`HybridCache`	**283.26**	**41.64**

D ShareGPT Results

We further evaluate prefix cache reuse on ShareGPT using Qwen3.5-35A3B with a 1K chunk size and tensor parallelism $T_p = 2$. As shown in Table 7, `HybridCache` reduces TTFT from 368.28 ms to 283.26 ms while improving the cache hit rate from 34.56% to 41.64%.

E Prompt Length and Chunk Size Statistics

Prompt lengths are measured with tokenizer without adding special tokens. DeepPlanning is measured from actual trace prompt-token counts across LLM calls, including system instructions and tool interaction history. Other datasets are measured from prompt fields shown in Table 8.

Table 8: Average prompt lengths and chunk sizes used in our experiments.

Benchmark	Number of Prompts	Avg. Prompt Tokens	Chunk Size
DeepPlanning	360	46,610	32,768
RULER-98K	260	97,249	32,768
ZebraLogic	1,000	895	256
LongProc	3,805	8,062	1,024

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

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

   Answer: [Yes]

   Justification: We have made the claims accurately reflect the contributions and scope of the paper.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

Justification: We have discussed the limitation of the work in Appendix A.

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: [Yes]

   Justification: The assumptions and complete proof are provided in Section 3.

   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 have discussed the implementation in Section 3 and the experiment setting in Section 4.

   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: [No]

   Justification: The current submission does not yet provide an open-access code release or a public reproducibility package. But we plan to release the code in a later version upon acceptance.

   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 have discussed the implementation in Section 3 and the experiment setting in Section 4.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: All efficiency experiments are deterministic. For accuracy experiments, we only provide the average results across five runs.

   Guidelines:

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification:We have discussed the experiment setting in Section 4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: To the best of our knowledge, the work conforms to 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]

    Justification: The paper discusses both potential positive impacts and misuse risks in Appendix B.

    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:The paper does not release a new model, dataset, or other asset that would require a dedicated safeguards section in the current submission.

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: [No]

Justification: The paper cites the main external datasets and models it uses, but it does not explicitly enumerate the license and terms of use for each asset in the current draft.

Guidelines:

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

13. **New assets**

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

Answer: [N/A]

Justification: The submission does not introduce or release a new dataset, codebase, or model artifact as a standalone asset

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: [No]

    Justification: The paper includes a small human annotation study for strategy-level validation, but it does not provide the full annotation instructions or any compensation details in the current submission.

    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: [No]

    Justification: The submission includes a human annotation component, but it does not currently discuss participant risks or IRB-equivalent review and approvals.

    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: LLMs are central to the paper. Our paper focus on improving the serving efficiency of LLM and all experiments are evaluated on most recent LLMs.

    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.