Strong Drafts Need Compact Memories: Long-Context Speculative Decoding with Compressed KV Cache

Tong Yuan, Chengxi Liao, Zeyi Wen*
Data Science and Analytics Thrust, Information Hub,
The Hong Kong University of Science and Technology (Guangzhou)
{tyuan053, cliao118}@connect.hkust-gz.edu.cn, wenzeyi@hkust-gz.edu.cn

Abstract

Long-context LLM applications such as document summarization and multi-turn agents require generation from prefixes spanning tens of thousands of tokens, making decoding latency a major bottleneck. Speculative decoding (SD) reduces latency without changing model outputs, but its speedup depends on both accepted draft tokens and draft-step latency: Lightweight drafts are fast but lack the capacity to capture long-range dependencies, whereas strong independent drafts recover acceptance but incur growing KV-access cost at long prefixes. We introduce memory-augmented drafting for long-context SD, equipping a strong independent draft with compressed draft-side KV memory: A lightweight adaptor constructs and incrementally updates this memory to retain distant information and exact recent context. The target verifier retains its full KV cache and applies the standard accept/reject rule, preserving SD's lossless guarantee. Experiments on Llama 3.1-8B and 70B targets at prefix lengths up to 32K show that our method reduces draft-side memory by over 70%. It achieves speedups of up to 2.08× and 3.33×, respectively, over autoregressive decoding.

1 Introduction

As LLM applications such as document summarization, online deep research, and multi-turn agents grow in scope and complexity, efficient generation becomes increasingly important (Khurana et al., 2023). However, the inherently sequential nature of Autoregressive decoding limits throughput and increases latency. Speculative decoding (SD) (Leviathan et al., 2023) reduces this sequential bottleneck without changing model outputs: a draft proposes several candidate tokens, and the target verifies them in parallel. SD speedup depends on both the number of accepted draft tokens and the latency of each draft step. Accepting more candidates amortizes each target verification over more output tokens, while a slow draft can erase that gain.

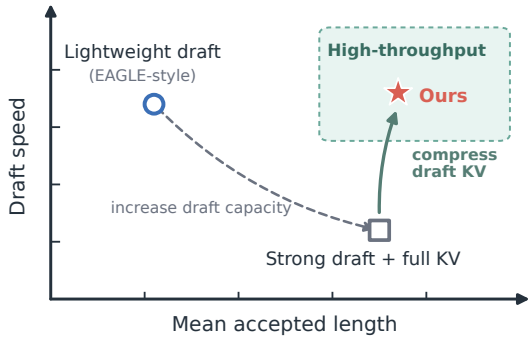


Figure 1: Lightweight drafts are fast but accept fewer tokens; strong full-KV drafts recover acceptance but incur high KV-access cost. Our compressed-memory draft targets high acceptance and speed.

When it comes to long-context generation, these applications can condition on prefixes spanning tens of thousands of tokens (Bai et al., 2024). Longer prefixes increase the cost of each decoding step: attention reads a larger KV cache and processes more key–value pairs, increasing both memory traffic and computation. Lightweight draft methods such as EAGLE (Li et al., 2024b) keep draft latency low over short prefixes (Figure 2, left). Their limited capacity, however, makes it difficult to capture long-range dependencies, so acceptance declines as the context grows (Chen et al., 2025; Bhendawade et al., 2025; Huo et al., 2026). Stronger independent drafts recover acceptance (Figure 2, middle), but every draft step streams the full historical KV cache, whereas KV traffic and computation grows with prefix length. At long prefixes, this KV cost dominates draft latency and erodes SD's end-to-end speedup. Figure 1 summarizes the resulting design target: a draft with sufficient capacity to capture long-range dependencies while maintaining high acceptance and low

*Corresponding author.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 6481–6494
October 24-29, 2026 ©2026 Association for Computational Linguistics

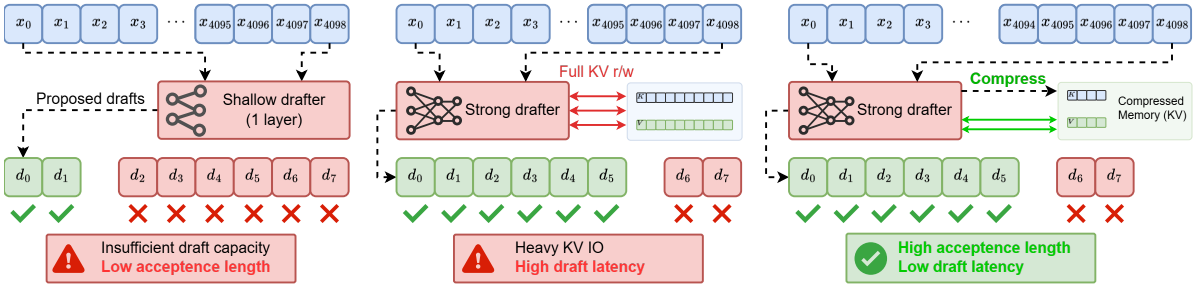


Figure 2: Overview of our proposed framework.

latency as the prefix grows.

Building on this analysis, we propose *memory-augmented drafting* for long-context SD with independent draft models (Figure 2, right). Instead of maintaining a full draft KV cache, the draft model keeps a compressed KV memory that carries prediction-relevant information from distant history while preserving exact recent context, thereby preserving both high acceptance and low draft latency (Figure 1). This memory is initialized during the prefill stage—which can run in parallel with target prefill since the draft is an independent model—and is incrementally updated as decoding proceeds. A lightweight memory adaptor produces the compressed states; only the adaptor parameters are trained while the draft backbone remains frozen, keeping training cost low. The target verifier retains its unmodified full KV cache, so the standard accept/reject rule applies and speculative decoding remains lossless. Experiments on Llama 3.1 family targets at prefix lengths up to 32K tokens show that memory-augmented drafting reduces draft-side memory by over 70% and achieves speedups of up to 2.08× on the 8B target and 3.33× on the 70B target, consistently outperforming both lightweight and full-KV draft baselines.

Our contributions are as follows:

- We identify the long-context SD dilemma: lightweight drafts lose acceptance as the prefix grows, while strong full-KV independent drafts introduce large overhead.
- We propose *memory-augmented drafting*, which equips independent draft models with compressed draft-side KV memory that preserves both high acceptance and low draft latency.
- The design is hardware-friendly: the compressed memory is continuous and append-only, so new slots are materialized and appended without in-place updates to existing entries, making it compatible with efficient KV-cache serving systems.
- We evaluate on Llama 3.1 family targets across long-input summarization benchmarks at prefix lengths up to 32K tokens, demonstrating consistent speedups over autoregressive decoding and speculative baselines while preserving the lossless guarantee of SD.

2 Related Work

2.1 Long-Context Speculative Decoding

Existing long-context SD methods progressively shift the bottleneck diagnosis from model compute to KV-cache access. TriForce (Sun et al., 2024) and MagicDec (Sadhukhan et al., 2025) drive the draft with StreamingLLM (Xiao et al., 2024) attention, dropping middle-range tokens to reduce KV traffic; TokenSwift (Wu et al., 2025) uses sparse Medusa heads within an end-to-end optimized decoding system; QuantSpec (Tiwari et al., 2025) quantizes the draft KV yet still scans the full prefix at every step; LongSpec (Yang et al., 2026) retrains an EAGLE-style draft with attention sinks (Xiao et al., 2024), retaining the shallow-layer capacity bottleneck. Collectively, these efforts show that compressing the draft-side KV cache is one of the most promising directions for long-context SD, yet none introduces a learnable compressed state to replace distant raw KV.

2.2 KV Cache Compression

Methods that compress the KV cache of a single model fall into three families. *Eviction-based* approaches (StreamingLLM (Xiao et al., 2024), LM-Infinite (Han et al., 2024), SnapKV (Li et al., 2024a), AdaKV (Feng et al., 2025)) discard selected positions, reducing the token count but creating a systematic mismatch when applied asymmetrically between draft and verifier. *Quantization-*

based approaches (KIVI (Liu et al., 2024), Chan-Mix (Liao and Wen, 2026)) trim memory bytes but still scan the full prefix length, and typically require custom GPU kernels for mixed-precision attention. *Context compression* (Gist (Mu et al., 2023), ICAE (Ge et al., 2024), AutoCompressor (Chevalier et al., 2023), Activation Beacon (Zhang et al., 2025)) replaces raw tokens with learnable summary tokens, producing continuous compressed representations that standard attention can consume without specialized kernels.

Among these families, training-aware context compression is most hardware-friendly because its outputs are dense, fixed-shape KV entries compatible with existing serving infrastructure.

3 Background & Motivation

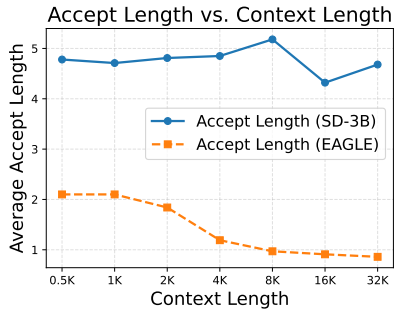
This section revisits the speculative decoding speedup model, then presents two empirical claims that together motivate our design.

3.1 Speculative Decoding

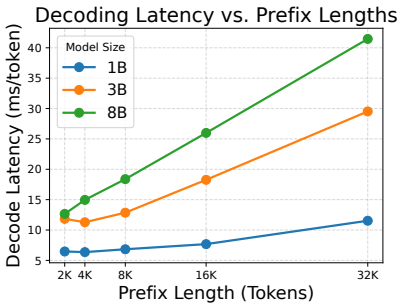
Speculative decoding accelerates autoregressive generation through a *draft–verify* paradigm that preserves the target model's output distribution (Leviathan et al., 2023). Given a target model M_t and a cheaper draft model M_d, each iteration first runs M_d for L_{draft} autoregressive steps to propose a candidate continuation, and then invokes M_t in a single parallel forward pass to verify the candidates. Let L_{acc} denote the mean number of accepted draft tokens per iteration. Standard SD emits these tokens plus one target token: a correction token after the first rejection or a bonus token when all candidates are accepted. Thus, ignoring early termination, the mean number of emitted tokens is $L_{\text{emit}} = L_{\text{acc}} + 1$. Let t_d and t_t denote the single-token latency of M_d and M_t, respectively. The decoding speedup over standard decoding is

$$\text{Speedup} = \frac{L_{\text{acc}} + 1}{L_{\text{draft}} \cdot \frac{t_d}{t_t} + 1}. \tag{1}$$

The $+1$ in the numerator counts the extra target token, whereas the $+1$ in the denominator represents one target verification pass normalized by t_t. Eq. 1 highlights that practical speedup is governed by two competing factors: a high acceptance length L_{acc}, which requires the draft to faithfully approximate the target, and a small relative draft cost t_d/t_t, which requires the draft to remain substantially cheaper than the target.



(a) Accepted draft length of EAGLE compared with the 3B draft model for Llama3.1-8B. Both methods use a 10 as draft length.



(b) Single token decode latency across prefix lengths and model sizes.

Figure 3: Performance and latency analysis across model capacity and context lengths.

3.2 Long-Context SD Needs a Stronger Draft Model

EAGLE-style methods (Li et al., 2024b) attach a single lightweight autoregressive layer to the target model. This design satisfies both terms of Eq. 1 on short prefixes: the shallow draft is cheap (t_d/t_t is small) and approximates the target well enough (L_{acc} is adequate). Once the prefix grows to tens of thousands of tokens, both properties degrade simultaneously. Figure 3a shows that EAGLE's L_{acc} drops monotonically as context length increases for Llama3.1-8B, while the independent Llama3.2-3B draft model maintains a high accepted length.

These experiments show that long-context SD requires capacity: the draft model must be strong enough to approximate the target over long histories. A lightweight single-layer draft cannot meet this requirement.

3.3 Context Length Dominates Latency

A stronger independent draft model recovers accepted length, but full-KV drafting requires reading the entire historical KV cache at every draft step. We decompose single-token decode latency

into a weight-related term and a KV-access term:

$$T_{\text{decode}} \approx \alpha d^2 + \beta L d, \tag{2}$$

where d is the hidden size and L is the prefix length. The first term reflects loading model parameters and is fixed for a given backbone; the second term reflects streaming the historical KV cache and grows linearly with L. Figure 3b confirms this decomposition: per-token decode latency scales nearly linearly in L, and the slope is dominated by KV access rather than model size.

Hence, the central bottleneck in long-context SD is not draft capacity alone, but the cost of conditioning a strong draft on long histories via full KV-cache access.

3.4 Summary

Eqs. 1 and 2 together reshape the design target for long-context SD (Figure 1). Because L_{acc} scales with draft capacity while t_d is dominated by KV access, an effective long-context draft should retain—or even increase—model capacity to keep L_{acc} high, yet access a substantially shorter KV memory to keep t_d/t_t low.

4 Method

Full-KV independent drafting streams the entire historical cache at every step, making draft latency grow linearly with context length; a pure sliding window cuts that cost but removes distant history from the draft, reducing acceptance. We propose *Memory-Augmented Sliding-Window (MASW) Drafting* (Figure 4), which equips an independent draft model with a compact working memory. MASW is designed to substantially reduce draft latency while preserving the accepted length of a strong independent draft. The target verifier keeps the standard full KV cache over all original tokens, so speculative decoding remains lossless. We define the three-part memory in §4.1, describe how slots are materialized in §4.2, and integrate the design into speculative decoding in §4.3.

4.1 From Full-KV Drafting to Memory-Augmented Sliding Window

An independent draft model that conditions on the full prefix must stream the entire historical KV cache at every draft step, so draft-side latency grows linearly with context length. The simplest reduction is a sliding window that retains only the most recent raw-token KV entries. A pure window, however, discards all information beyond the frontier, which misaligns the draft with the target's full-prefix conditionals and hurts acceptance.

We augment the window with *memory slots*: compact states materialized at a fixed rate from completed history and kept in the draft cache after their raw KV is released. At step t, the draft-side working memory is

$$\mathcal{M}_t = \mathcal{M}_{\text{sink}} \cup \mathcal{M}_{\text{local},t} \cup \mathcal{M}_{\text{slot},t}, \tag{3}$$

Following the sink-plus-local-window structure of StreamingLLM (Xiao et al., 2024), let $\mathcal{S} = \{1, \ldots, S\}$ denote the first S raw-token positions. Their KV entries form $\mathcal{M}_{\text{sink}}$ (red, left in Figure 4) and remain in the draft cache throughout prefill and decoding. We call $S = |\mathcal{S}|$ the *sink-token count*. The local memory $\mathcal{M}_{\text{local},t}$ stores the exact raw KV of the most recent W non-sink tokens (blue, right in Figure 4), where W is the *local-window width*:

$$\mathcal{M}_{\text{local},t} = \big\{(K_i, V_i) \mid \max(S{+}1,\, t{-}W{+}1) \le i \le t\big\},$$

MASW augments this structure with $\mathcal{M}_{\text{slot},t}$, which stores multi-layer KV for memory slots (yellow, interleaved with blue tokens in Figure 4) that carry distant context. During materialization, previous slots carry compressed earlier history into the new slot. Because the draft accesses only S sink entries, W local entries, and one slot per r tokens, the working-set size scales with S, W, and $\lfloor t/r \rfloor$ rather than the full prefix length t. The target verifier maintains the usual full KV over all original tokens and is unchanged by this compression.

4.2 Memory Materialization

At each compression boundary $\tau_m = mr$, which occurs every r tokens (excluding sink tokens), the draft inserts a memory-slot token g_m (the orange position in Figure 4) and runs it through the same transformer blocks as ordinary tokens. We define r as the *slot interval*: the draft materializes one memory slot for every r raw tokens. This gives a nominal $r\times$ compression ratio for the materialized history, but not for the complete draft cache, which also retains sink tokens and the local window.

$$g_m \leftarrow f_\theta\big(\mathcal{M}_{\text{local},\tau_m},\ \mathcal{M}_{\text{sink}},\ \mathcal{M}_{\text{slot},<m}\big),$$

where f_θ is the masked forward pass through all transformer layers, and θ denotes the frozen draft backbone together with the trainable memory-adaptor parameters. Each slot acts as a *compressed*

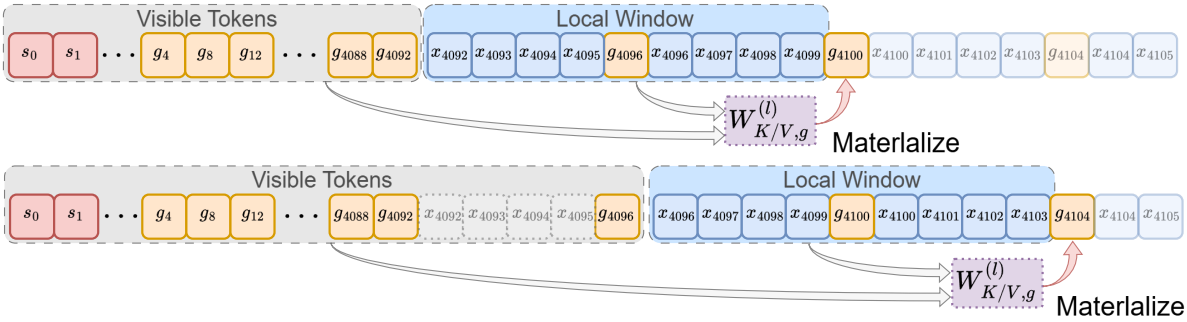


Figure 4: Memory-augmented sliding-window drafting at two consecutive compression boundaries. **Top:** the draft materializes slot g_{4100} at the current boundary. **Bottom:** the window has advanced by one slot interval (after 4 decoding steps); the oldest raw KV is evicted, the previously materialized slot is retained as memory, and the next slot g_{4104} is produced at the new boundary.

checkpoint of the current window while reading prior slots, so successive slots form an incremental compression chain rather than disjoint segment summaries.

Structured attention as write policy. A structured attention mask, not a hand-designed pooling rule, realizes this update. After raw tokens are evicted, later tokens can access their information only through the corresponding memory slots. The next-token objective therefore trains each slot to functionally replace those tokens for subsequent prediction, rather than summarize the document or reconstruct individual KV entries.

Projection and KV materialization. At every Transformer layer l, memory-slot hidden states use dedicated *mirrored projection matrices* $W_{K,g}^{(l)}$ and $W_{V,g}^{(l)}$ to produce their keys and values:

$$K_r^{(l)} = H_r^{(l-1)} W_{K,r}^{(l)}, \quad V_r^{(l)} = H_r^{(l-1)} W_{V,r}^{(l)},$$
$$K_g^{(l)} = H_g^{(l-1)} W_{K,g}^{(l)}, \quad V_g^{(l)} = H_g^{(l-1)} W_{V,g}^{(l)}.$$

Only memory-slot states use $W_{*,g}^{(l)}$; raw-token states continue to use the original projections $W_{*,r}^{(l)}$. The original projections remain frozen, while the mirrored matrices are initialized from the corresponding backbone weights and constitute the memory adaptor's trainable parameters. This separate projection branch decouples the information aggregation performed inside the slot hidden state from the KV entries that the slot writes into the cache. For each layer l, the materialized KV pair $(K_{g_m}^{(l)}, V_{g_m}^{(l)})$ is appended to $\mathcal{M}_{\text{slot},t}$ during the draft forward pass. In the same pass, raw-token KV outside the retained rollback window is evicted; this update does not wait for target verification. Restricting trainable parameters to $W_{*,g}^{(l)}$ keeps the adaptor lightweight and leaves the frozen draft weights untouched, so the draft retains its standalone decoding behavior; we revisit this scope in the Limitations.

Memory update cycle. As illustrated in Figure 4, the draft model periodically materializes a memory slot every r tokens by compressing the current local window into long-term KV states. After materialization, raw KV outside the sliding window is discarded while its information remains accessible through the accumulated memory slots. This materialization-and-eviction cycle repeats throughout decoding. Each slot g_m shares the RoPE position ID of the immediately following raw token. The raw token keeps its original position ID, so slot insertion creates a duplicate position ID rather than shifting subsequent raw-token positions.

4.3 Speculative Decoding with MASW

MASW compresses only the draft-side KV; the target verifier retains its full-prefix KV and applies standard parallel verification and rejection sampling. Thus, memory quality affects accepted length and speed, while target verification preserves the output distribution.

Prefill: The target model prefills the full prefix to build its complete KV cache. The draft model processes the same prefix in one forward pass to construct its initial $\mathcal{M}_t$; because the draft is an independent model, its prefill can run in parallel with the target prefill. This draft-side prefill uses the structured memory mask described in Appendix C, so memory slots are materialized during prefix processing without exposing ordinary draft tokens to the full raw-token history. Slot KV is produced and cached within masked draft forward passes, with

no separate retrieval or compression pass. Prefix slots are created during prefill, and new slots are materialized at compression boundaries during decoding. Subsequent speculative iterations read only the compact draft memory while verification uses the full cache.

Rollback: When the target rejects a speculative block, the draft discards speculative local KV and any memory slots created after the last accepted position. The retained raw-KV rollback window recovers the active local context after rejection.

5 Evaluation

We evaluate MASW on long-context speculative decoding tasks spanning two target-model scales (Llama 3.1-8B and 70B) and prefix lengths from 8K to 32K tokens. §5.1 describes the experimental setup; §5.2 reports decoding speedups against autoregressive and speculative baselines; §5.3 analyzes draft-side efficiency and the effect of draft prefill on first-token latency; and §5.4 ablates key design choices.

5.1 Experimental Setup

Models and hardware. We use Llama 3.1-8B-Instruct and Llama 3.1-70B-Instruct (Grattafiori et al., 2024) as target models, and pair them with Llama 3.2-3B-Instruct and Llama 3.1-8B-Instruct as independent draft backbones. MASW equips each draft backbone with the mirrored projection matrices described in §4.2; the backbone remains frozen and only the memory-adaptor parameters are trained. All inference measurements run on a single 8×H100 (80 GB) node, and the main latency evaluation uses batch size 1. For heterogeneous batches, per-sequence attention masks disable the extra slot position for sequences that are not at a materialization boundary. This masking leaves the target KV cache and lossless verification unchanged.

MASW configuration. We set the local-window width and sink-token count to $W = S = 128$, giving every memory slot the same number of visible raw tokens. MASW uses a 16-token raw-KV rollback window. We train separate adaptors for nominal 4× and 8× compression.

Training. We train the mirrored projection matrices $W^{(l)}_{*,g}$ in two stages. For pretraining, we sample 2B tokens from RedPajama (Weber et al., 2024) and append an end-of-sequence delimiter to each document. Supervised fine-tuning (SFT) uses task-oriented long-context data from LongAlpaca (Chen et al., 2024) and BookSum (Kryscinski et al., 2022), with all sequences truncated to 8K tokens. The training objective minimizes next-token prediction loss over raw tokens conditioned on the compressed draft memory $\mathcal{D}_t = (\mathcal{M}_{\text{sink}}, \mathcal{M}_{\text{local},t}, \mathcal{M}_{\text{slot},t})$:

$$\mathcal{L} = \sum_t -\log\, p(x_t \mid \mathcal{D}_t,\, x_{<t}).$$

Memory-slot positions are excluded from the loss because they serve as compressed carriers rather than prediction targets.

Datasets. We evaluate on mixed summarization tasks, including GovReport (Huang et al., 2021), QMSum (Zhong et al., 2021), and MultiNews (Fabbri et al., 2019) from LongBench-v1 (Bai et al., 2024). These tasks require the model to attend to information distributed across the full prefix rather than concentrated at the boundaries.

Baselines. We compare against four categories of methods: (1) vanilla autoregressive (AR) decoding; (2) standard speculative decoding (SD) with uncompressed draft KV; (3) EAGLE (Li et al., 2024b) and EAGLE-3 (Li et al., 2025), the leading short-context SD methods that use a single lightweight autoregressive layer on top of the target; (4) component-level draft-side KV reduction baselines using sliding-window attention (SWA) and SnapKV (Li et al., 2024a).

Metrics. We report three indicators: the mean number of tokens emitted per speculative iteration (*Tok./Iter*), decoding throughput (*tok/s*), and decoding speedup over the AR baseline (*Speedup*). All main experiments and ablations use greedy decoding with temperature T=0. Appendix D provides table-specific configurations, detailed baseline implementations, and evaluation protocols.

5.2 Main Results

Table 1 compares all methods across two target scales and prefix lengths from 8K up to 32K. EAGLE and EAGLE-3 provide little or inconsistent acceleration at long contexts, confirming the capacity bottleneck identified in §3. SWA is particularly fragile because discarding distant KV sharply reduces accepted length. Full-KV SD also loses speedup as the prefix grows because draft-side KV traffic increases with context length. MASW avoids this decline. Its strongest settings gain speedup with context on the 8B target and remain consistently strong on the 70B target, where the larger verifier amortizes the compressed draft cost.

		8K			16K			24K			32K		
		Tok./Iter	tok/s	Speedup	Tok./Iter	tok/s	Speedup	Tok./Iter	tok/s	Speedup	Tok./Iter	tok/s	Speedup
L3.1-8B	AutoRegressive	1.00	46.15	1.00×	1.00	30.40	1.00×	1.00	22.64	1.00×	1.00	18.21	1.00×
	EAGLE	2.42	49.64	1.08×	1.33	17.97	0.59×	2.36	23.75	1.05×	2.34	18.94	1.04×
	EAGLE-3	2.07	42.46	0.92×	2.07	29.27	0.96×	2.06	22.75	1.01×	2.06	19.24	1.06×
	SWA	2.33	42.67	0.92×	1.49	11.32	0.37×	1.40	11.12	0.49×	1.29	9.40	0.52×
	SnapKV	3.66	41.70	0.90×	3.08	31.21	1.03×	3.65	33.06	1.46×	2.88	23.84	1.31×
	SD L3B	5.18	54.33	1.18×	4.32	29.94	0.98×	4.12	20.50	0.91×	4.68	17.89	0.98×
	Ours 4× L3B	4.79	57.42	1.24×	4.22	44.50	1.46×	4.35	41.92	1.85×	4.72	37.91	2.08×
	Ours 8× L3B	3.46	41.48	0.90×	3.47	36.61	1.20×	3.40	34.22	1.51×	3.14	26.87	1.48×
	Ours 4× L8B	4.94	53.64	1.16×	4.81	46.74	1.54×	4.99	42.64	1.88×	3.64	27.62	1.52×
	Ours 8× L8B	5.51	59.83	1.30×	5.34	51.85	1.71×	5.35	46.59	2.06×	4.42	35.38	1.94×
L3.1-70B	AutoRegressive	1.00	10.09	1.00×	1.00	7.38	1.00×	1.00	5.82	1.00×	1.00	4.79	1.00×
	EAGLE	1.60	6.46	0.64×	1.61	5.05	0.69×	1.47	3.89	0.67×	1.58	3.70	0.77×
	EAGLE-3	1.26	5.08	0.50×	1.04	3.26	0.44×	1.06	2.81	0.48×	1.06	2.48	0.52×
	SWA	3.03	5.99	0.59×	1.86	3.43	0.47×	1.99	3.57	0.61×	1.87	3.04	0.63×
	SnapKV	4.45	9.76	0.97×	3.84	7.76	1.05×	3.90	7.33	1.26×	3.72	6.61	1.38×
	SD L3B	4.31	24.85	2.46×	4.14	16.97	2.30×	3.57	10.94	1.88×	4.05	9.76	2.03×
	SD L8B	4.78	22.96	2.28×	4.65	15.59	2.11×	4.35	11.02	1.89×	4.82	9.98	2.08×
	Ours 4× L3B	4.36	27.49	2.73×	3.88	19.74	2.68×	3.60	15.53	2.67×	3.65	13.13	2.74×
	Ours 8× L3B	3.72	23.46	2.33×	3.84	19.54	2.65×	3.58	16.04	2.75×	3.43	12.32	2.57×
	Ours 4× L8B	4.91	29.14	2.89×	4.99	24.54	3.33×	4.07	16.93	2.91×	3.78	13.25	2.76×
	Ours 8× L8B	5.15	30.54	3.03×	4.16	20.46	2.77×	3.78	15.73	2.70×	3.77	13.54	2.82×

Table 1: Main long-input speculative decoding comparison across two target models and four prefix lengths. *Tok./Iter* denotes the mean number of tokens emitted per speculative iteration and does not apply to AutoRegressive decoding. *tok/s* denotes decoding throughput, and *Speedup* denotes decoding speedup relative to the AutoRegressive baseline. For each target model and prefix length, the best *Speedup* and the corresponding Tok./Iter and tok/s entries are shaded.

Across draft backbones, 4× compression tends to preserve more tokens per iteration, whereas 8× further reduces draft latency, making the best ratio dependent on target scale and prefix length.

5.3 Effectiveness & Efficiency Analysis

We analyze MASW's draft-side effectiveness and efficiency in terms of prefill latency, decoding latency, and peak memory usage.

Prefill latency. MASW materializes prefix memory during draft-side prefill, which adds computation. However, Table 2 shows that the memory-augmented 8B draft still prefills faster than the 70B target at all tested prefix lengths.

Model	8K	16K	32K
8B draft, original	306.5	711.2	1835.2
8B draft, with memory	751.6	2350.2	7992.5
70B target, original	2212.9	5480.3	12316.4

Table 2: Prefill latency (ms) across prefix lengths.

Let P_d and P_t denote draft and target prefill latency, respectively. Because both prefills run concurrently,

$$T_{\text{prefill}}^{\text{MASW}} = \max(P_d, P_t) = P_t \quad (P_d < P_t). \quad (4)$$

Thus, the additional work is hidden behind target prefill and does not increase TTFT in this setting.

Draft	Weights	32K Extra Peak	Tok./Iter	Latency (ms)
3B		17.21 GB	4.05	41.34
Ours 3B 4×	5.98 GB	4.42 GB	3.65	13.94
Ours 3B 8×		3.98 GB	3.43	12.39
8B		18.02 GB	4.82	54.92
Ours 8B 4×	14.95 GB	5.05 GB	3.78	15.37
Ours 8B 8×		4.55 GB	3.77	14.02

Table 3: Draft-side effectiveness and efficiency under the L3.1-70B target at 32K prefix length. *Weights* reports frozen draft-backbone weight memory. *Extra Peak* reports all peak GPU memory beyond the backbone weights, including memory-adaptor weights when present, the KV cache, attention masks, logits, and temporary buffers.

Draft-side efficiency. Table 3 compares full-KV and MASW drafting at a 32K prefix under the L3.1-70B target. MASW reduces extra peak memory from 17.21–18.02 GB to 3.98–5.05 GB and draft latency from 41.34–54.92 ms to 12.39–15.37 ms. Meanwhile, Tok./Iter decreases only from 4.05 to 3.43 for the 3B draft and from 4.82 to approximately 3.78 for the 8B draft. Increasing compression from 4× to 8× provides further memory and latency savings while preserving Tok./Iter on the 8B draft.

5.4 Ablation Study

We ablate four design choices in MASW: local-window size, mirrored-projection initialization, training procedure, and training-context length.

5.4.1 Local-Window Size

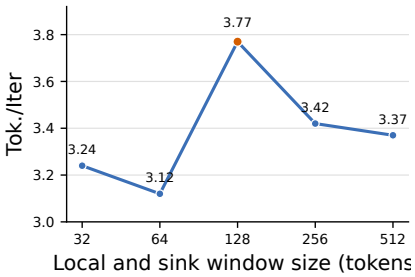


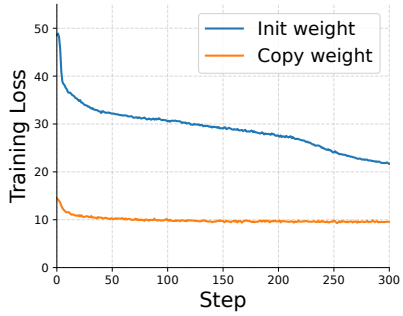
Figure 5: Window-size ablation.

We vary the local-window size from 32 to 512 tokens for the L3.1-8B draft at nominal 8× compression, using the L3.1-70B target and a 32K prefix. For each configuration, we set the sink-token count equal to the local-window width ($S = W$) so that every memory slot observes the same number of raw local tokens. Figure 5 shows that accepted length remains relatively stable across the tested range and peaks at 3.77 with a 128-token window. We therefore use $W = S = 128$ in the main experiments.

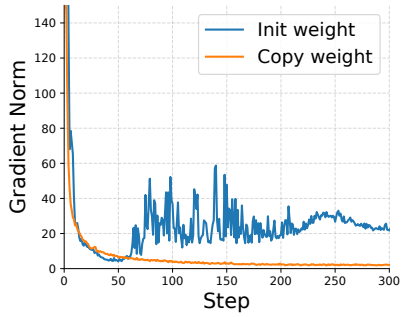
5.4.2 Mirrored Projection Initialization

We compare two strategies for initializing the mirrored projection matrices $W^{(l)}_{*,g}$: random initialization versus copying from the backbone's pretrained KV projections $W^{(l)}_{*,r}$. We pretrain the L3.2-3B memory adaptor at nominal 4× compression under each strategy.

Figure 6 shows a persistent gap between the two strategies. Random initialization starts at a much higher loss and, even after 300 steps, never reaches the range that copy-weight initialization already occupies at step 0. The gradient norm tells the same story: random initialization stays elevated and noisy throughout training, whereas copy-weight initialization decays into a small, stable regime within the first tens of steps. Preserving the backbone's KV geometry gives the optimizer a strong starting point, so we adopt copy-weight initialization as the default throughout all experiments.



(a) Pretraining loss.



(b) Gradient norm.

Figure 6: Pretraining loss and gradient norm for the L3.2-3B memory adaptor at nominal 4× compression.

Model	**Ratio**	PT only	SFT only	PT + SFT
L3B	4×	3.70	3.23	**4.15**
	8×	3.78	3.24	**4.78**
L8B	4×	4.22	3.18	**4.81**
	8×	3.87	3.01	**5.34**

Table 4: Mean number of tokens emitted per speculative iteration (Tok./Iter) at 16K context under three training recipes. **Bold** marks the best recipe per row.

5.4.3 Training Recipe

We compare three training recipes for the memory adaptor: continued *pretraining* alone, *SFT* alone, and the two-stage *pretrain-then-SFT* pipeline. Table 4 reports mean Tok./Iter at 16K context across both draft backbones and nominal compression ratios.

PT + SFT achieves the highest Tok./Iter across all settings. SFT alone is consistently weaker, suggesting that supervised data cannot by itself teach the adaptor to form robust compressed representations. Pretraining provides a stronger starting point, but still needs SFT to align the memory slots with the downstream drafting distribution. The advantage of the two-stage recipe becomes more pronounced under nominal 8× compression, where the adaptor must preserve more information in fewer

slots. We therefore use pretrain-then-SFT as the default recipe.

5.4.4 Training-Context Length

Training Context	8K	16K	24K	32K
8K	**5.15**	**4.16**	**3.78**	**3.77**
32K	4.96	4.01	3.45	3.58

Table 5: Mean number of tokens emitted per speculative iteration (Tok./Iter) for L3.1-8B adaptors trained with 8K or 32K contexts at nominal 8× compression.

We train the L3.1-8B adaptor at nominal 8× compression with either 8K- or 32K-token sequences, using a fixed 2B-token training budget, and evaluate both adaptors with the L3.1-70B target. As shown in Table 5, the adaptor trained on 8K sequences achieves higher Tok./Iter at every evaluation length, including 32K. Rather than learning a representation tied to an absolute context length, MASW learns a relative slot-materialization operation: each slot combines the bounded local window, sink tokens, and earlier slots, and the same operation is repeatedly applied as the context grows. Under a fixed token budget, 8K sequences provide more training instances, which likely improves the data efficiency of learning this operation. These results show that the learned compression extrapolates beyond the training context length and that longer training sequences are not necessary in this setting.

6 Conclusion

We introduced Memory-Augmented Sliding-Window (MASW) Drafting, which replaces the full draft KV with a compact working memory of sink tokens, an exact local window, and periodically materialized memory slots produced by trainable mirrored projection matrices. Across long-context settings, this design substantially reduces draft-side memory and latency while preserving much of the acceptance quality of full-KV drafting. Because the target verifier operates on the unmodified full KV cache, the lossless guarantee of speculative decoding is preserved. Lower latency and serving cost may also facilitate undesirable uses of existing language models, such as large-scale generation of misleading, abusive, or otherwise harmful content.

Limitations

Because training resources are limited, we train the memory adaptor on only 2B tokens with an 8K context length. This leaves open a broader study of adaptor-training configurations, including longer contexts, larger training corpora, and full fine-tuning of the draft model so that it can learn context compression more natively. Training only the mirrored K/V projections keeps the adaptor lightweight, yet may cap how much information each slot can encode; jointly fine-tuning a larger subset of draft parameters, such as MLP blocks or low-rank updates on the backbone, is a natural extension we leave to future work. MASW currently fixes the local-window size, sink-token count, and slot interval to obtain regular masks, stable cache layouts, and bounded materialization overhead; adaptive allocation remains future work. MASW could also compress the draft path in self-speculative decoding. However, using the target model or part of it for long-context drafting may weaken cost-effectiveness; we leave this trade-off to future work.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (NSFC) under Grant No. 62306256 and the Natural Science Foundation of Guangdong Province under Grant No. 2025A1515010261.

AI Assistance We used AI assistants for preliminary information retrieval and organization, code debugging, and grammar-level checking and polishing. All retrieved information, suggested references, code changes, and textual edits were manually checked and verified by the authors.

References

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

Nikhil Bhendawade, Irina Belousova, Qichen Fu, Henry Mason, Antonie Lin, Mohammad Rastegari, and Mahyar Najibi. 2025. Speculative Streaming: Efficient and scalable speculative decoding with multi-stream attention. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 19536–19559, Suzhou, China. Association for Computational Linguistics.

Guanzheng Chen, Qilong Feng, Jinjie Ni, Xin Li, and Michael Qizhe Shieh. 2025. RAPID: Long-context inference with retrieval-augmented speculative decoding. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 8093–8107. PMLR.

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2024. LongLoRA: Efficient fine-tuning of long-context large language models. In *The Twelfth International Conference on Learning Representations*.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. 2023. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3829–3846, Singapore. Association for Computational Linguistics.

Alexander Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir Radev. 2019. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1074–1084, Florence, Italy. Association for Computational Linguistics.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. 2025. Ada-KV: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference. In *Advances in Neural Information Processing Systems*, volume 38, pages 113152–113188. Curran Associates, Inc.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. 2024. In-context autoencoder for context compression in a large language model. In *The Twelfth International Conference on Learning Representations*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 540 others. 2024. The Llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2024. LM-infinite: Zero-shot extreme length generalization for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3991–4008, Mexico City, Mexico. Association for Computational Linguistics.

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

Feiye Huo, Jianchao Tan, Jiahao Liu, Zixu Jiang, Jiacheng Li, Jingang Wang, Xunliang Cai, and Shengli Sun. 2026. RepSpec: Structural re-parameterized draft model training for speculative decoding. In *The Fourteenth International Conference on Learning Representations*.

Diksha Khurana, Aditya Koli, Kiran Khatter, and Sukhdev Singh. 2023. Natural language processing: state of the art, current trends and challenges. *Multimedia Tools and Applications*, 82(3):3713–3744.

Wojciech Kryscinski, Nazneen Rajani, Divyansh Agarwal, Caiming Xiong, and Dragomir Radev. 2022. BOOKSUM: A collection of datasets for long-form narrative summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 6536–6558, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024a. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, volume 37, pages 22947–22970. Curran Associates, Inc.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024b. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 28935–28948. PMLR.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2025. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc.

Chengxi Liao and Zeyi Wen. 2026. Channel-Aware Mixed-Precision Quantization for Efficient Long-Context Inference. In *The Fourteenth International Conference on Learning Representations*.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 32332–32344. PMLR.

Jesse Mu, Xiang Li, and Noah Goodman. 2023. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems*, volume 36, pages 19327–19352. Curran Associates, Inc.

Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian En-Hsu Yen, Avner May, Tianqi Chen, and Beidi Chen. 2025. MagicDec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. In *The Thirteenth International Conference on Learning Representations*.

Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. 2024. TriForce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. In *First Conference on Language Modeling*.

Rishabh Tiwari, Haocheng Xi, Aditya Tomar, Coleman Richard Charles Hooper, Sehoon Kim, Maxwell Horton, Mahyar Najibi, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. 2025. QuantSpec: Self-speculative decoding with hierarchical quantized KV cache. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 59668–59686. PMLR.

Maurice Weber, Daniel Y. Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, Ben Athiwaratkun, Rahul Chalamala, Kezhen Chen, Max Ryabinin, Tri Dao, Percy Liang, Christopher Ré, Irina Rish, and Ce Zhang. 2024. RedPajama: An open dataset for training large language models. In *Advances in Neural Information Processing Systems*, volume 37.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Tong Wu, Junzhe Shen, Zixia Jia, Yuxuan Wang, and Zilong Zheng. 2025. TokenSwift: Lossless acceleration of ultra long sequence generation. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 67650–67669. PMLR.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*.

Penghui Yang, Cunxiao Du, Fengzhuo Zhang, Haonan Wang, Tianyu Pang, Chao Du, and Bo An. 2026. LongSpec: Long-context lossless speculative decoding with efficient drafting and verification. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1826–1844, San Diego, California, United States. Association for Computational Linguistics.

Peitian Zhang, Zheng Liu, Shitao Xiao, Ninglu Shao, Qiwei Ye, and Zhicheng Dou. 2025. Long context compression with activation beacon. In *The Thirteenth International Conference on Learning Representations*.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir Radev. 2021. QMSum: A new benchmark for query-based multi-domain meeting summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5905–5921, Online. Association for Computational Linguistics.

A Scientific Artifacts

This work uses existing scientific artifacts, including publicly available research models and datasets, solely for research and evaluation purposes. We cite the original creators of the models, datasets, baselines, and software frameworks used in our experiments in the corresponding experimental setup and baseline sections, as well as in the references. We use these artifacts consistently with their stated intended uses and access conditions, and we follow the licenses or terms specified by their original providers.

We do not collect new human-subject data, annotate private data, or redistribute raw datasets. The datasets used in our experiments are standard public research benchmarks. To the best of our knowledge, based on their original documentation, these datasets are not intended to contain personally identifying information. Since our work focuses on efficiency evaluation rather than dataset construction, we rely on documentation from the original artifact creators for details such as data source, domain, language coverage, and usage restrictions. We also document the models, datasets, evaluation settings, and implementation details used in our experiments.

B Potential Risks

MASW accelerates inference but does not add new content-generation capabilities, and it inherits the safety, bias, and privacy risks of its underlying models and training data. Greater inference efficiency may nevertheless enable harmful uses of existing models at a larger scale. The lossless guarantee also relies on verifying every draft proposal with the target model; deployments that weaken or bypass this verification may change model outputs. Practitioners should therefore retain full target verification and apply the same access controls, monitoring, and content safeguards required for the underlying model.

C Memory Prefill

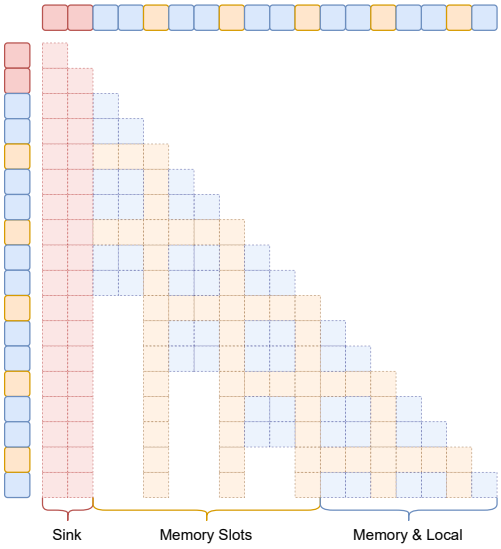


Figure 7: Memory prefill attention mask.

During draft-side prefill, the attention mask enforces the same memory-write policy used during decoding. Sink tokens remain globally visible to stabilize attention, and each memory-slot token can attend to previously materialized slots together with the raw tokens in its assigned local unit and neighboring boundary tokens. Ordinary raw tokens, in contrast, are restricted to the compact memory view: sink tokens, available memory slots, and the bounded local window. This asymmetric visibility lets the prefill pass build the initial compressed memory $\mathcal{M}_t$ while avoiding a full raw-token KV cache for the draft model. Figure 7 illustrates this mask structure.

D Detailed Experimental Setting

Component-level baselines. We reproduce SWA and SnapKV within the same Hugging Face Transformers framework (Wolf et al., 2020). For both baselines, the draft and target use the same model backbone and weights. The draft generates tokens autoregressively using SWA or SnapKV for draft-side KV reduction, whereas verification uses the full target KV cache. SWA uses a sliding-window size of 1,024 tokens. SnapKV retains 4,096 KV entries and otherwise follows its official default configuration. For EAGLE and EAGLE-3, we use chain-based proposal generation with the released checkpoints and the remaining official inference settings. All speculative methods in Table 1 propose five draft tokens per iteration; the separate

motivation experiment in Figure 3a uses a 10-token proposal horizon.

Decoding and timing. The reported 8K, 16K, 24K, and 32K lengths refer to the input prefix before generation and exclude output tokens. We filter examples from GovReport, QMSum, and MultiNews whose tokenized inputs match the evaluated prefix length. Each run generates at most 1,024 output tokens and stops early when the model emits an end-of-sequence token. *Tok./Iter* denotes the mean number of tokens emitted per speculative iteration. Decoding throughput (*tok/s*) is the number of generated output tokens divided by decoding wall-clock time; draft and target prefill are excluded. *Speedup* denotes decoding speedup over the corresponding AutoRegressive baseline under the same prefix length and timing scope.

E Notations

Part	Notation	Explanation
Speculative decoding	M_d, M_t	draft and target models
	L_{draft}	number of draft tokens proposed per iteration; fixed to 5
	L_{acc}	mean number of accepted draft tokens per iteration
	L_{emit}	mean emitted tokens per nonterminal iteration; reported as Tok./Iter
	t_d, t_t	single-token latency of the draft and target models
Draft memory	$\mathcal{D}_t$	compact draft-side working memory at step t
	$\mathcal{M}_{\text{sink}}$	exact KV for the first S raw tokens
	$\mathcal{M}_{\text{local},t}$	exact raw-token KV in the local window
	$\mathcal{M}_{\text{slot},t}$	materialized KV entries for memory slots
	$\mathcal{S} = \{1, \ldots, S\}$	sink-token positions
	S	sink-token count
	W	local-window width
	r	slot interval; one slot per r raw tokens
Memory slots	g_m	memory-slot token for unit m
	$\tau_m = mr$	compression boundary for slot g_m
	$\mathcal{A}(g_m)$	write scope of memory-slot token g_m
	$\mathcal{G}_{<m}$, U_m, $\mathcal{N}_m$	earlier slots, current unit, and neighboring tokens
Transformer and KV	d	hidden size
	L	prefix length in the latency model
	$K_r^{(l)}$, $V_r^{(l)}$	raw-token key and value at layer l
	$K_g^{(l)}$, $V_g^{(l)}$	memory-slot key and value at layer l
	$W_{K,r}^{(l)}$, $W_{V,r}^{(l)}$	frozen backbone KV projections for raw tokens
	$W_{K,g}^{(l)}$, $W_{V,g}^{(l)}$	trainable mirrored KV projections for memory slots
	$W_{*,r}^{(l)}$, $W_{*,g}^{(l)}$	shorthand for raw-token and memory-slot KV projections

Table 6: List of notation used in this paper.