# Adaptive Mass-Segmented KV Compression for Long-Form Reasoning

**Junzhe Yang**[1,2], **Xiaoyu Shen**[2]*

[1] Shanghai Jiao Tong University
[2] EIT-NLP Lab, Eastern Institute of Technology, Ningbo
yang_j@sjtu.edu.cn, xyshen@eitech.edu.cn

## Abstract

The linear growth of the Key-Value (KV) cache is a critical bottleneck in long-form LLM inference. Existing KV compression methods mitigate this by evicting tokens based on importance scores. However, we show that their reliance on global Top-$k$ selection triggers *Region Wipe-out*: the severe eviction of contiguous reasoning blocks that derails logical coherence. To address this, we propose Adaptive Mass-Segmented (AMS) KV Compression, a framework that shifts the paradigm from token-level competition to region-aware quota allocation. AMS adaptively partitions the KV cache based on the spatial distribution of attention mass, ensuring structurally vital reasoning segments receive guaranteed memory quotas. To ensure stability during iterative decoding, an EMA-based smoothing mechanism is incorporated to prevent jitter in segment boundaries. Crucially, AMS is a universal plug-and-play layer that is orthogonal to existing scorers. It can be seamlessly integrated into representative methods such as TOVA, Expected Attention, KeyDiff, R-KV and TriAttention. We further validate a vLLM-style paged-KV compaction path that preserves head-wise KV selection and supports standard compact-cache attention. Production-serving efficiency remains outside the scope of our measurements. Extensive experiments across a diverse suite of tasks, including mathematical reasoning (MATH500, AIME, GSM8K), code completion, open-domain QA, and sparse retrieval, demonstrate that AMS consistently mitigates structural fragmentation and boosts model performance. Code is available at: https://github.com/EIT-NLP/AdaptiveMassSegment

## 1 Introduction

While LLMs excel at complex tasks, the linear growth of the KV cache has become a major bottleneck in inference [1–3]. This issue is particularly pronounced in long-form reasoning, where generating extended Chain-of-Thought (CoT) sequences continuously expands the cache [4–7]. As decoding is fundamentally IO-bound, repeatedly loading these growing KV states from HBM creates a memory wall that significantly degrades latency [2, 8].

To mitigate this cost, decoding-time KV compression offers a practical mechanism for bounded memory inference [1, 9, 10]. These methods periodically compress the KV cache during generation by performing on-the-fly eviction of less important entries [11–23]. At each compression step, tokens are scored based on importance metrics such as accumulated or expected attention [1, 9, 10, 14, 24–26], geometric variation [7, 27], or reasoning-aware criteria [5, 6]. Despite the diversity in metrics, these approaches often rely on a *budgeted token-level competitive* paradigm, where individual tokens across the entire context compete globally for a fixed number of slots via Top-$k$ selection [1, 10, 24, 26–29].

*Corresponding author.

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

However, we find that this token-level competition introduces a critical structural weakness under aggressive compression. Top-$k$ selection naturally favors highly salient "heavy hitter" tokens [1, 30], while distributed but contextually important reasoning spans receive insufficient protection. As a result, entire intermediate reasoning regions can be removed from the cache, a failure mode we term *Region Wipe-out*. Related forms of KV fragmentation and context loss have also been observed in prior work [28, 31]. Region Wipe-out breaks the continuity of reasoning trajectories and introduces temporal gaps that impair logical consistency [32], leading to several characteristic behaviors such as *(1) Problem Drifting*, where the model silently alters key constraints; *(2) Premature Overturning*, where valid intermediate conclusions collapse after supporting context is removed; and *(3) Repetition Collapse*, where severe context starvation drives the model into degenerate loops [33]. [2] Several prior works attempt to reduce fragmentation through chunk- or block-based compression [22, 25, 31]. While such approaches partially alleviate direct token competition, their partition structures remain largely fixed and cannot adapt to the evolving distribution of reasoning importance. Consequently, they often fail to adequately protect dense reasoning spans.

We propose Adaptive Mass-Segmented (AMS) KV compression, a framework that reformulates KV compression from *token-level direct competition* into *region-aware quota allocation*. Instead of using importance scores solely as individual survival signals, AMS interprets them as a global spatial distribution over the reasoning trajectory. Our framework first constructs adaptive segments based on the distribution of attention mass, then allocates retention quotas with minimum preservation guarantees for each segment before performing local token selection. This *allocate-then-score* paradigm explicitly shields structurally important regions from being entirely wiped out. Moreover, because decoding-time compression is repeatedly applied throughout generation, allocation decisions based purely on instantaneous attention signals can fluctuate across compression events [26, 29, 34]. To reduce quota jitter across compression events, AMS introduces a lightweight history-aware regional mass estimation mechanism based on exponential moving average (EMA) smoothing [26, 29], applying temporal stabilization at the allocation level rather than modifying the base scorer. [3]

Extensive experiments demonstrate the effectiveness of AMS across diverse reasoning and generation tasks, including mathematical reasoning (MATH500, AIME, GSM8K), code completion, open-domain question answering, and sparse retrieval. Notably, AMS delivers its strongest gains under constrained KV budgets: on MATH500, AMS-Expected improves over AdaKV-ExpE2 by up to $+16.0$ points, surpasses uncompressed Full KV at $T_{\text{keep}} = 512$ and $1024$, and further boosts the strong TriAttention scorer by up to $+6.4$ points. A key advantage of AMS lies in its modularity: rather than replacing existing token importance estimators, it serves as an orthogonal allocation layer that can be seamlessly integrated into existing KV compression pipelines such as TOVA [10], Expected Attention [24], TriAttention [7], KeyDiff [27], and R-KV [5], consistently improving their performance without architectural modifications (Table 5). Beyond accuracy, AMS also exhibits strong systems practicality. Unlike masking-based methods such as AdaKV-ExpE2 that retain the full KV cache in memory, AMS follows a gather-and-compact design that maintains the same physical cache footprint as standard gather-based compressors while achieving substantially stronger performance. AMS is naturally compatible with vLLM-style paged KV serving: the selector produces head-wise keep indices, while the runtime materializes them through paged-KV compaction and block-table replacement, allowing standard compact-cache attention without additional steady-state attention overhead (Appendix E). Empirically, AMS matches or improves the latency of existing gather-based policies, while also reducing repetition collapse during free-form reasoning, leading to faster end-to-end generation under the same KV budget (Appendix H.2).

Our contributions are as follows: (1) We identify *Region Wipe-out* as a fundamental failure mode in KV cache compression, revealing how token-level Top-$k$ selection disrupts long-form reasoning under tight memory budgets. (2) We propose *Adaptive Mass-Segmented (AMS) KV Compression*, a region-aware allocate-then-score framework that combines adaptive segmentation, segment-wise quota allocation, and EMA-based stabilization to preserve coherent reasoning trajectories during repeated compression. (3) We show that AMS serves as a general plug-and-play allocation layer that can be seamlessly integrated into diverse KV compression methods, consistently improving both effectiveness and robustness under constrained memory budgets. (4) We characterize the memory footprint, single-request latency, and periodic recompression overhead of AMS, and validate its compatibility with vLLM paged-KV compaction path.

[2] These phenomena are illustrated in Figure 2 and further analyzed in Appendix C.

[3] As shown in Table 7, removing EMA credit degrades accuracy, especially at larger cache budgets.

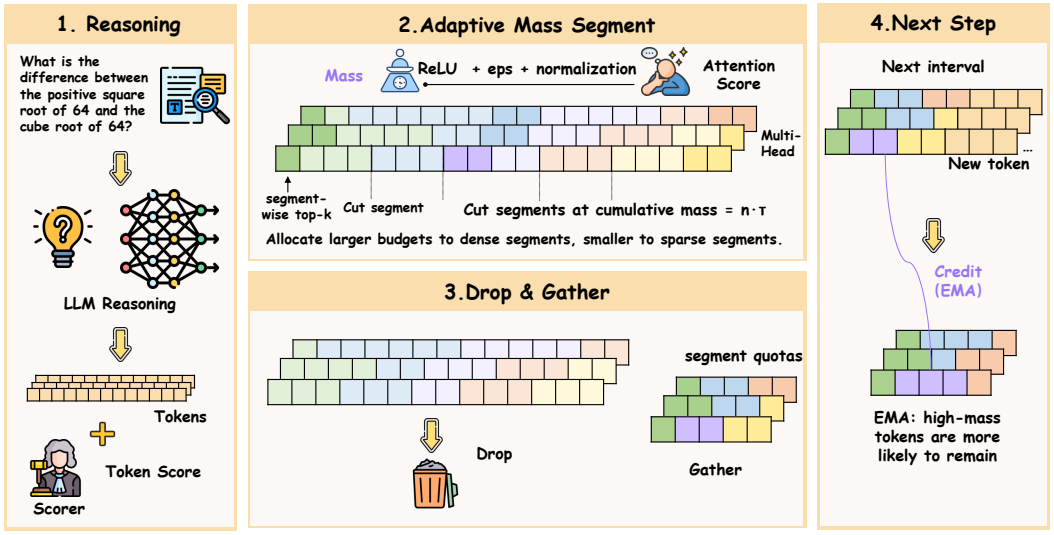



**Figure 1: Overview of AMS for decoding-time KV compression.** (1) Given a long chain-of-thought reasoning process, the base model generates tokens and a training-free scorer assigns token-level importance scores. (2) AMS converts recent attention-based usage into a normalized quality mass, partitions the sequence into adaptive segments based on cumulative mass, and allocates larger quotas to dense segments and smaller quotas to sparse ones. (3) Within each segment, AMS performs segment-wise top-$k$ selection under the segment quota, drops evicted tokens, and gathers the retained KV entries into a compact cache. (4) At the next compression event, AMS reuses an EMA-based credit signal that accumulates past mass, making consistently high-mass tokens more likely to remain and stabilizing selection across events.

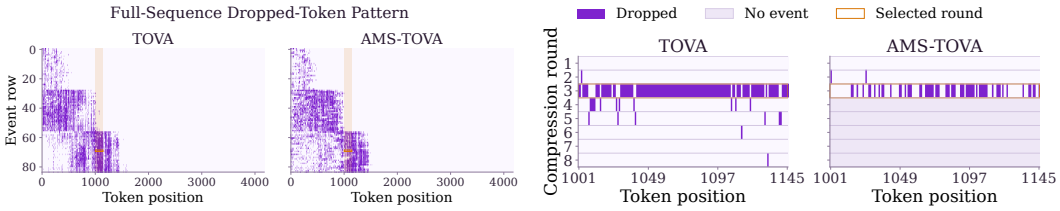
## 2 Related Work

**Decoding-time KV eviction and training-free scoring.** Decoding-time KV compression bounds memory by evicting cached tokens online, typically combining locality with globally important tokens such as sinks or heavy hitters [1, 9, 35]. Many training-free methods score tokens by attention, reconstruction, geometry, or reasoning-aware proxies, including TOVA, KeyDiff, SnapKV, expected-attention scoring, TriAttention, R-KV, and RPC [5–7, 10, 18, 24, 27]. However, retention is still often implemented as global or per-head top-$k$ selection, which lacks explicit protection for contiguous temporal regions and can be brittle under tight budgets and repeated recompression [26, 28, 29].

**Adaptive budgeting and structure-aware compression.** Prior work allocates KV budgets non-uniformly across layers or heads, such as PyramidKV, OmniKV, LaCache, AdaKV, DuoAttention, HeadKV, and RazorAttention [11–17]. Other methods organize caches into chunks, blocks, clusters, prototypes, or trees, including ChunkKV, SABlock, ClusterKV, ProtoKV, TreeKV, and HeteroCache [19–23, 25]. These approaches improve budget allocation across layers, heads, or coarse cache units, but they do not explicitly allocate quotas to adaptive temporal regions within each head. In contrast, AMS introduces *attention-derived adaptive segments* along the sequence and assigns explicit region-wise quotas before token-level scoring.

**Stability and systems.** Repeated recompression can destabilize context usage, motivating history-aware scoring such as ReST-KV and G-KV [26, 29]. AMS instead makes the *allocation* history-aware, using EMA credit to smooth adaptive segments and quotas while keeping the scorer unchanged. Recent long-output reasoning systems such as LongFlow, Lethe, and ThinKV improve KV efficiency through fused eviction, layer/time-adaptive pruning, or thought-level quantization–eviction; AMS is orthogonal, providing scorer-agnostic within-head temporal quota allocation [36–38]. [4]

[4] We discuss paged-KV compatibility with vLLM [2] in Appendix E.




**(a)** Full-sequence view.

**(b)** Zoomed context.

**Figure 2: Motivating dropped-token burst example.** Purple pixels denote dropped tokens. The full-sequence view highlights a local region where TOVA forms a dense contiguous dropped-token burst. The zoomed view shows the same token window across compression rounds: AMS-TOVA fragments the dropped positions in the selected round, illustrating a local failure mode that motivates adaptive segment-wise allocation.

# 3 Method

## 3.1 Fundamental Setting

Figure 1 gives a high-level overview of the AMS pipeline. We follow the standard decoding-time KV compression paradigm: during autoregressive generation, a Transformer decoder periodically compresses its KV cache from length $T$ to a fixed target budget $T_{\text{keep}}$. We compute retention policies independently for each attention head. As illustrated in Figure 2(a), traditional token-wise Top-$k$ policies (e.g., TOVA) tend to over-index on local attention spikes. To mitigate this, AMS replaces global Top-$k$ selection with region-wise quotas (Figure 2(b)), distributing retention more evenly across the temporal history without altering the backbone or the underlying scoring function.

At each compression event, AMS keeps the backbone and scoring function unchanged, but modifies the selection stage by (i) constructing adaptive segments along the sequence within each KV head (ii) allocating token quotas to these segments under the fixed cache budget, and (iii) using a temporally smoothed mass signal to stabilize allocation. We next describe these components in more detail.

## 3.2 Head-wise Quality Mass from Attention

At the core of our framework is a head-wise *quality mass* distribution, which shifts attention signals from a token survival metric to a macro-level spatial partitioning tool. At a compression event, we collect per-position usage scores $u_i$ derived from recent-window attention over the last $W$ decoding queries (detailed implementations, including sequence pooling and causal mask padding, are deferred to Appendix B.4).[5] We then map these scores to a non-negative normalized mass distribution:

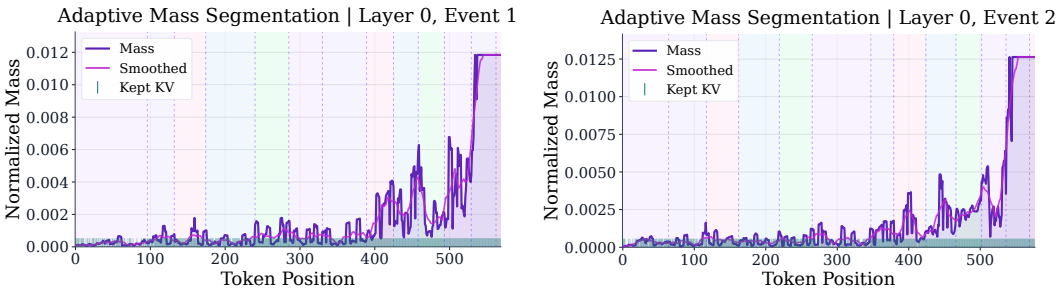
$$m_i = \frac{\max(u_i, 0) + \epsilon}{\sum_{v=1}^{T} (\max(u_v, 0) + \epsilon)}, \tag{1}$$

where $\epsilon > 0$ is a small constant. The resulting mass vector $m \in \mathbb{R}^T$ satisfies $\sum_{i=1}^{T} m_i = 1$. By design, $m$ assigns larger weights to regions with dense attention usage. This mass distribution subsequently drives our adaptive segmentation and quota allocation, entirely decoupled from the final token-level retention choices.

## 3.3 Adaptive Mass-Segmented KV Compression

**Mass-based adaptive segmentation.** Given the quality-mass curve along the sequence, we partition each head's cache into adaptive segments before allocating the budget. Figure 3 visualizes the normalized quality mass and the resulting adaptive segments at two consecutive compression events: regions with higher mass are cut into finer segments, while low-mass regions are merged into longer ones. By enforcing budget allocation at the segment level and guaranteeing that each segment retains at least a few tokens, we ensure that no temporal region is completely removed from the compressed cache, so every part of the history keeps at least a minimal representation.

[5] Attention-derived token-importance signals are widely used in training-free KV-cache eviction, including attention sinks, heavy-hitter retention, token omission, and observation-window-based KV selection [1, 9, 10, 18].




**(a)** Compression event 1.

**(b)** Compression event 2.

**Figure 3: Quality mass and adaptive segmentation.** The solid curve shows normalized quality mass over current KV-cache positions, not absolute generation positions. Shaded bands and dashed lines denote adaptive segments, and teal ticks mark retained KV positions. High-mass regions form finer segments under a fixed $T_{\text{keep}}$.

Consider a single KV head with mass vector $m \in \mathbb{R}^T$ satisfying $\sum_{t=1}^{T} m_t = 1$. We first compute the prefix-sum curve $c(t) = \sum_{u=1}^{t} m_u$ for $t = 1, \ldots, T$. Given a target segment mass $\Delta =$ `segment_mass`, we define cumulative-mass thresholds $\Delta, 2\Delta, 3\Delta, \ldots$ and, for each threshold $k\Delta$, find the smallest position $b_k = \min\{\, t \mid c(t) \geq k\Delta \,\}$. Geometrically, this process maps the sequence into a 1D space where segment boundaries are dynamically determined by attention-mass density, fulfilling our macro-level partitioning criterion. These cut points $\{b_k\}$, together with the start and end positions, induce an initial segmentation into intervals $[a_i, b_i)$ that each contain roughly $\Delta$ total mass.

The raw cuts may produce segments that are too short or too long in terms of token count. We therefore constrain segment lengths to lie in a reasonable range by applying simple split and merge heuristics: segments longer than a length threshold $L_{\max}$ are split into roughly equal-sized subsegments, while neighboring segments shorter than $L_{\min}$ are merged. This yields a set of non-overlapping segments $\{[a_i, b_i)\}_i$ that cover all token positions $\{1, \ldots, T\}$, cutting high-mass regions more finely and grouping low-mass regions more coarsely in preparation for segment-wise quota allocation.

**Segment-wise Quota Allocation.** Once the sequence has been partitioned into segments, we allocate the limited retention budget across segments before any token-level selection. This allocation layer enforces simple structural guarantees: every segment keeps at least a few tokens, high-mass regions receive more capacity, and globally important positions are always preserved, directly mitigating the Region Wipe-out phenomenon identified in Section 1.

Consider a single KV head with segments $\{[a_i, b_i)\}_{i=1}^{S}$ and mass vector $m$. We define the mass and length of segment $i$ as $M_i = \sum_{t=a_i}^{b_i-1} m_t$ and $L_i = b_i - a_i$. Given a global keep budget $T_{\text{keep}}$ and a per-segment minimum $q_{\min}$, we first assign each segment a minimum quota $q_i^{\min} = \min(q_{\min},\, L_i)$, which guarantees that no segment is completely erased as long as the global budget is large enough. The remaining budget $T_{\text{rem}} = T_{\text{keep}} - \sum_{i=1}^{S} q_i^{\min}$ is then distributed to segments in proportion to their mass as $q_i = q_i^{\min} + \text{round}(T_{\text{rem}} \cdot \frac{M_i}{\sum_{j=1}^{S} M_j})$, followed by a lightweight adjustment step to enforce $\sum_i q_i = T_{\text{keep}}$ and $q_i \leq L_i$. This yields a segment-wise quota vector $\{q_i\}$ that respects both the global budget and per-segment length constraints.

We additionally incorporate global must-keep rules that are standard in practice. A small number of sink tokens at the beginning of the sequence and a suffix of the most recent tokens are always retained; these positions form a must-keep set $\mathcal{I}_{\text{must}}$ and take precedence when forming the final index set. If necessary, segment quotas are slightly downscaled so that the union of $\mathcal{I}_{\text{must}}$ and segment-level selections fits into the overall budget. In our implementation, must-keep positions are inserted first and the remaining budget is allocated to segments; if the remaining budget becomes negative, we reduce the recent-token suffix while always preserving sink tokens.

**In-Segment Selection with a Plug-and-Play Scorer.** Completing our "allocate-then-score" paradigm, after segment-wise quotas have been determined, we perform token selection locally within each segment. This step is strictly decoupled from budget allocation: AMS only assumes

access to a base scorer that assigns a real-valued importance score to each cached position, and uses the segment-wise quotas to constrain how many tokens can be kept from each region.

Formally, let $g \in \mathbb{R}^{B \times H_{kv} \times T}$ denote scores returned by a base scorer such as a TOVA-style press or an expected-attention-based scorer. For a given batch element and KV head, and for each segment $[a_i, b_i)$ with quota $q_i$, we select the top-$q_i$ indices within the segment according to $g$. This yields a set of candidate indices $\mathcal{I}_{\text{seg}} = \bigcup_i \text{TopK}\big(\{a_i, \ldots, b_i - 1\},\, q_i;\, g\big)$, which is then combined with the global must-keep set $\mathcal{I}_{\text{must}}$ from the previous step.

We form the final keep set by taking the union $\mathcal{I} = \mathcal{I}_{\text{seg}} \cup \mathcal{I}_{\text{must}}$, removing duplicates, and, if necessary, trimming or backfilling a small number of positions to match the target budget $T_{\text{keep}}$. Trimming is performed by discarding the lowest-scoring non-must-keep positions, while backfilling (when $|\mathcal{I}| < T_{\text{keep}}$ due to short segments) adds high-scoring positions outside the current segment quotas. Finally, we apply $\mathcal{I}$ to gather the compressed cache along the sequence dimension as $K' = K[:, :, \mathcal{I}, :]$ and $V' = V[:, :, \mathcal{I}, :]$, and continue decoding with $(K', V')$. Because this selection step only requires a score tensor of shape $[B, H_{kv}, T]$, AMS can wrap a wide range of existing scoring rules, and the mass-segmented allocation core remains compatible with diverse KV eviction methods.

**EMA Credit for Temporal Stability.** To stabilize quota allocations across repeated compression events, we introduce an optional exponential-moving-average (EMA) credit mechanism that makes the mass signal history-aware before segmentation and quota allocation[6].

For each layer and KV head, we maintain a credit vector $c \in \mathbb{R}^T$ that accumulates past mass assignments. At a new compression event, given the current mass $m^{(e)}$, we update $c \leftarrow \lambda\, c + (1 - \lambda)\, m^{(e)}$, where $\lambda \in (0, 1)$ is a decay parameter. We then normalize the credit into a distribution and form a history-aware mass $m_{\text{used}} = \text{normalize}\big(\beta\, m^{(e)} + (1 - \beta)\, \text{normalize}(c)\big)$, with mixing coefficient $\beta \in [0, 1]$. The resulting $m_{\text{used}}$ replaces $m$ in the segmentation and quota allocation steps; all subsequent logic is unchanged. Intuitively, tokens and regions that remain consistently useful across multiple events accumulate higher credit and therefore receive a larger share of the budget over time, while transient spikes in recent attention are smoothed out. In practice, this history-aware mass leads to smaller changes in segment boundaries and keep sets across compression events (Sec. 4.5).

# 4 Experiments

## 4.1 Experimental Settings

**Benchmarks and models.** We evaluate decoding-time KV compression primarily on mathematical reasoning (MATH500 [39, 40], AIME24, AIME25 [41], and GSM8K [42]), with additional long-context workloads (e.g., code completion and open-domain QA) evaluated in Section 4.4. Our main backbone is DEEPSEEK-R1-DISTILL-QWEN-7B [43], with scalability tested on its 32B variant [44] and cross-backbone generalization on OPENTHINKER3-7B [45]. All methods are evaluated under identical decoding-time constraints (e.g., per-layer budget $T_{\text{keep}} \in \{256, 512, 1024\}$, interval $I = 512$). We compare AMS against representative training-free KV-cache compressors, including StreamingLLM [9], TOVA [10], KeyDiff [27], ChunkKV-Expected [25], PyramidKV [11], AdaKV-ExpE2 [14, 24], R-KV [5], RPC [6], and TriAttention [7]. Main results instantiate AMS with TOVA and expected-attention scorers, yielding AMS-TOVA and AMS-Expected. Additional plug-in results, including TriAttention, KeyDiff, and R-KV, are reported in Table 5. Detailed evaluation protocols, baseline configurations, and AMS hyperparameters are deferred to Appendix B.

## 4.2 Main Results

**Overall performance.** We evaluate decoding-time KV compression on MATH500, AIME24, and AIME25 using DEEPSEEK-R1-DISTILL-QWEN-7B as the backbone. Table 1 reports pass@1 accuracy for the main compared methods under per-layer cache budgets $T_{\text{keep}} \in \{256, 512, 1024\}$. The gains across different baseline philosophies are substantial. On MATH500, **AMS-TOVA** improves over the token-wise **TOVA** by $+7.2$, $+3.8$, and $+4.6$ absolute points at $T_{\text{keep}} = 256, 512, 1024$,

[6] History-aware smoothing has been used to stabilize KV-cache eviction under repeated compression [26, 29].

**Table 1:** Pass@1 accuracy (%) under decoding-time KV compression with per-layer effective cache length $T_{\text{keep}} \in \{256, 512, 1024\}$. Baselines are grouped by their underlying compression paradigm. AMS-TOVA uses TOVA as the base scorer, while AMS-Expected uses the expected-attention scorer. We discuss their absolute improvements over the corresponding baselines in the main text. Additional plug-in results with other scorers are provided in Table 5. AMS variants are placed immediately below their corresponding base scorers for paired visual comparison. Bold numbers denote the best compressed result in each column.

| Method | MATH500 | | | AIME24 | | | AIME25 | | |
|---|---|---|---|---|---|---|---|---|---|
| | 256 | 512 | 1024 | 256 | 512 | 1024 | 256 | 512 | 1024 |
| Full KV | | 52.80 | | | 33.33 | | | 30.00 | |
| ***Token-level Eviction*** | | | | | | | | | |
| StreamingLLM | 28.00 | 41.20 | 49.60 | 0.00 | 0.00 | 3.33 | 0.00 | 3.33 | 13.33 |
| TOVA | 29.20 | 44.60 | 48.80 | 0.00 | 6.67 | 16.67 | 0.00 | 3.33 | 13.33 |
| ↪ AMS-TOVA | 36.40 | 48.40 | 53.40 | 0.00 | 6.67 | **20.00** | **3.33** | 13.33 | 20.00 |
| KeyDiff | 22.80 | 35.60 | 49.40 | 0.00 | 0.00 | 0.00 | 0.00 | 3.33 | 13.33 |
| ***Structure-aware Allocation (Layer / Head / Chunk)*** | | | | | | | | | |
| ChunkKV-Expected | 37.20 | 48.20 | 53.00 | 0.00 | 3.33 | 10.00 | 0.00 | 6.67 | 16.67 |
| PyramidKV | 31.60 | 42.20 | 51.80 | 0.00 | 0.00 | **20.00** | **3.33** | 3.33 | 6.67 |
| AdaKV-ExpE2 | 32.60 | 46.60 | 53.40 | 0.00 | 6.67 | 10.00 | 0.00 | 6.67 | 26.67 |
| ↪ AMS-Expected | **48.60** | **54.00** | **54.20** | **6.67** | **20.00** | 16.67 | 0.00 | **23.33** | **33.33** |
| ***Reasoning-specific Compression*** | | | | | | | | | |
| R-KV | 37.20 | 46.40 | 52.40 | 0.00 | 6.67 | 10.00 | **3.33** | 10.00 | 23.33 |
| RPC | 34.60 | 47.20 | 52.60 | 0.00 | 6.67 | 10.00 | **3.33** | 10.00 | 23.33 |

respectively. The expected-attention version **AMS-Expected** is also strong, outperforming the corresponding expected-attention baseline **AdaKV-ExpE2** by $+16.0$, $+7.4$, and $+0.8$ points at these budgets. These results indicate that the proposed allocation layer improves both token-wise and expected-attention-based scoring under the same decoding-time constraints. Additional plug-in results with geometric and reasoning-specific scorers, including TriAttention, are detailed in Table 5.

Full KV serves as an uncompressed reference rather than a theoretical upper bound on task accuracy. Prior work has reported that selective KV-cache compression can match or surpass Full KV in certain settings [46, 47]. One possible explanation is that removing low-utility cached states can mitigate attention dilution and change the distribution of attention over the remaining context [47].

**Universality across diverse base scorers.** To rigorously validate AMS as a universal plug-and-play layer, we further integrate it with geometric, gradient-free, and reasoning-specific scoring paradigms. While Table 1 focuses on overall comparisons, Table 5 isolates the absolute gains brought specifically by our allocation layer on MATH500. As shown, AMS yields consistent improvements across these diverse scorers. Most notably, under the extreme budget of $T_{\text{keep}} = 256$, AMS boosts KeyDiff from $22.80\%$ to $42.80\%$ (+20.00 points) and R-KV from $37.20\%$ to $42.60\%$, demonstrating the broad efficacy of decoupling region allocation from token scoring. As shown in Table 4, AMS's structural protection scales efficiently, yielding significant gains on DEEPSEEK-R1-DISTILL-QWEN-32B.

**Effect of KV budget.** As the keep budget shrinks, the gap between AMS and the baselines widens. On MATH500, for example, **StreamingLLM** and **TOVA** achieve $28.0\%$ and $29.2\%$ pass@1 at $T_{\text{keep}} = 256$, whereas **AMS-Expected** attains $48.6\%$ at the same budget. This pattern suggests that explicit region-wise budget allocation is particularly beneficial under aggressive decoding-time compression, allowing AMS to recover much of the accuracy lost by strong token-wise baselines when the KV cache is severely constrained. We attribute this widening gap under tight budgets to the region-level minimum guarantees in AMS, which prevent any temporal span from being fully erased in a single recompression event and thus preserve intermediate reasoning states needed by later steps.

**Results on GSM8K.** We also evaluate AMS on GSM8K, which consists of shorter word problems with natural-language solutions, using the same DEEPSEEK-R1-DISTILL-QWEN-7B backbone and the same decoding-time KV-budget setting. Table 2 reports pass@1 accuracy at target cache sizes $\{64, 128\}$. At both budgets, **AMS-Expected** achieves the best performance among the training-free

**Table 2:** Pass@1 accuracy (%) on GSM8K with DEEPSEEK-R1-DISTILL-QWEN-7B.

| $T_{\text{keep}}$ | Streaming LLM | TOVA | AdaKV-ExpE2 | AMS-Expected |
|---|---|---|---|---|
| 64 | 46.7 | 48.9 | 53.5 | **56.7** |
| 128 | 61.8 | 57.1 | 62.9 | **65.3** |

**Table 3:** Pass@1 accuracy (%) on MATH500 with OPENTHINKER3-7B.

| $T_{\text{keep}}$ | Streaming LLM | TOVA | AMS-TOVA | AMS-Expected |
|---|---|---|---|---|
| 128 | 9.2 | 6.4 | 12.2 | **23.6** |
| 512 | 36.8 | 25.8 | 37.4 | **45.4** |

**Table 4: Scalability across Model Scales.** Pass@1 accuracy (%) on MATH500(QWEN-32B).

| Method | QWEN-32B ($T_{\text{keep}}$) | | |
|---|---|---|---|
| | 256 | 512 | 1024 |
| Full KV | | 47.80 | |
| StreamingLLM | 11.60 | 24.20 | 40.00 |
| TOVA | 11.40 | 25.80 | 41.40 |
| AdaKV-ExpE2 | 16.20 | 28.80 | 40.40 |
| R-KV | 26.40 | 35.60 | 40.60 |
| RPC | 20.80 | 36.00 | 46.40 |
| AMS-TOVA (Ours) | 20.60 | 33.40 | 44.00 |
| AMS-Expected (Ours) | **33.00** | **43.00** | **47.00** |

**Table 5: Universal Plug-in Enhancements.** Pass@1 (%) on MATH500 using 7B backbone. AMS decouples allocation from scoring, significantly boosting diverse base scorers.

| Method | MATH500 ($T_{\text{keep}}$) | | |
|---|---|---|---|
| | 256 | 512 | 1024 |
| TriAttention (Geo.) | 55.00 | 56.80 | 57.20 |
| **+ AMS-TriAtt.** | **56.20**$^{\uparrow 1.20}$ | **60.60**$^{\uparrow 3.80}$ | **63.60**$^{\uparrow 6.40}$ |
| KeyDiff (Grad-free) | 22.80 | 35.60 | 49.40 |
| **+ AMS-KeyDiff** | **42.80**$^{\uparrow 20.00}$ | **53.20**$^{\uparrow 17.60}$ | **60.40**$^{\uparrow 11.00}$ |
| R-KV (Reasoning) | 37.20 | 46.40 | 52.40 |
| **+ AMS-R-KV** | **42.60**$^{\uparrow 5.40}$ | **53.60**$^{\uparrow 7.20}$ | **60.80**$^{\uparrow 8.40}$ |

baselines: it reaches $56.7\%$ at target cache size $64$ and $65.3\%$ at $128$, outperforming the token-wise **TOVA** baseline by $+7.8$ and $+8.2$ points, respectively. Compared to the head-adaptive **AdaKV-ExpE2**, AMS-Expected still yields consistent gains ($+3.2$ at $64$ and $+2.4$ at $128$), while retaining the practical advantage of gather-based KV compression under a fixed physical cache budget. Overall, these results indicate that region-wise allocation within each head can also improve robustness on more natural word-problem settings once the KV budget is moderately large.

### 4.3 Cross-Backbone Results

To test whether AMS generalizes beyond our primary Qwen-style backbone, we repeat decoding-time KV compression experiments on OPENTHINKER3-7B. We keep the evaluation protocol identical to Section 4.1: the same compression schedule, must-keep rules (sink and recent tokens), and the same AMS hyperparameters tuned on the Qwen backbone are reused without any re-tuning. We report pass@1 on MATH500 under two representative cache budgets, $T_{\text{keep}} \in \{128, 512\}$.

Table 3 shows that AMS improves robustness on OPENTHINKER3-7B. At $T_{\text{keep}} = 512$, **AMS-Expected** reaches $45.4\%$ pass@1, outperforming **StreamingLLM** ($36.8\%$) and **TOVA** ($25.8\%$), and coming within $0.4$ points of the no-compression reference ($45.8\%$). At $T_{\text{keep}} = 128$, token-wise baselines degrade sharply: **StreamingLLM** drops to $9.2\%$ and **TOVA** to $6.4\%$, while **AMS-TOVA** reaches $12.2\%$ and **AMS-Expected** reaches $23.6\%$. These results suggest that AMS transfers well across 7B-class reasoning backbones under identical decoding-time constraints.

### 4.4 Generality and System Efficiency

As shown in Table 6, AMS is not limited to math derivations. It achieves competitive performance across code, QA, and crucially, maintains strong overall capabilities on LONGBENCH and sparse retrieval (NIAH). Notably, AMS-Expected outperforms all baselines and even the uncompressed Full KV on code completion (27.44%), while AMS-TOVA achieves the highest compressed score on TRIVIAQA (70.88%). Furthermore, we evaluate the system-level memory and latency overhead of our proposed framework (Table 8). Importantly, AMS is compatible with paged-KV frameworks like vLLM, supporting gather-and-compact execution without steady-state attention overhead, and the serving runtime directly handles the underlying block-table replacements (Appendix E). As shown in Table 9, AMS effectively suppresses logical deadlocks and repetitive hallucinations in generated text, often resulting in faster end-to-end wall-clock completion despite its minor computational

**Table 6: Generality Beyond Mathematical Reasoning.** Performance across code completion, QA, LONGBENCH, and NIAH. AMS explicitly preserves diverse structural context.

| Method | Repo | TriviaQA | LongBench | NIAH |
|---|---|---|---|---|
| Full KV (No Comp.) | 24.04 | 69.34 | 19.30 | 0.3751 |
| StreamingLLM | 26.92 | 64.57 | 19.19 | 0.2113 |
| TOVA | 22.40 | 67.68 | 19.12 | 0.1894 |
| PyramidKV | 24.10 | 65.65 | 19.02 | 0.1777 |
| AdaKV-ExpE2 | 19.12 | 43.54 | 17.58 | 0.0866 |
| AMS-TOVA (Ours) | 23.64 | **70.88** | **19.30** | **0.2384** |
| AMS-Exp. (Ours) | **27.44** | 69.47 | 19.25 | 0.2209 |

**Table 7: Ablation of AMS components** on MATH500. We report pass@1 (%) under decoding-time KV compression with a TOVA scorer. Red denotes absolute performance drops.

| Variant | Target Cache ($T_{\text{keep}}$) | |
|---|---|---|
| | 512 | 1024 |
| **AMS (full)** | **48.4** | **53.4** |
| w/o mass-weighted quotas | 46.0$\downarrow$2.4 | 52.8$\downarrow$0.6 |
| w/o EMA credit | 48.0$\downarrow$0.4 | 50.8$\downarrow$2.6 |
| fixed-length segments | 48.0$\downarrow$0.4 | 50.4$\downarrow$3.0 |
| global head-only top-$k$ | 42.8$\downarrow$5.6 | 49.2$\downarrow$4.2 |

**Table 8: Memory and latency efficiency.** Peak memory and average decoding time on MATH500.

| Method | Mem. (GB) $\downarrow$ | | Time (s) $\downarrow$ | |
|---|---|---|---|---|
| | 512 | 1024 | 128 | 512 |
| StreamingLLM | **15.0** | **14.9** | 50.1 | 48.4 |
| TOVA | **15.0** | 14.9 | 67.1 | 52.3 |
| PyramidKV | 15.3 | 15.6 | 67.5 | 51.6 |
| AdaKV-ExpE2 | 39.6 | 39.7 | 91.8 | 62.3 |
| AMS-Expected | **15.0** | 14.9 | **41.4** | **44.0** |

**Table 9:** Free-generation dynamics. AMS mitigates repetition collapse, reducing the repetition rate by 5.4% compared to TOVA. This prevents degenerate loops, decreasing total generated tokens and speeding up end-to-end runtime.

| Method | Tokens $\downarrow$ | Rep. (%) $\downarrow$ | Time (s) $\downarrow$ |
|---|---|---|---|
| TOVA | 2963.6 | 21.58 | 67.34 |
| AMS-TOVA | **2799.7** | **16.18** | **63.85** |

overhead. Detailed quantitative system profiling is provided in Appendix H. We also defer the detailed mechanistic diagnostics, such as visualizing the mitigation of Region Wipe-out, to Appendix C.

### 4.5 Ablation Studies

We ablate AMS components on MATH500 using **DeepSeek-R1-Distill-Qwen-7B**. Following the setup in Section 4.2, we report pass@1 under decoding-time KV compression. As shown in Table 7, our design choices are crucial for maintaining reasoning performance.

AMS separates scoring from budget allocation. To verify that the gains come from explicit region-wise structural protection rather than a new scoring rule, we first replace region-wise quotas with a single global head-only top-$k$ selection. This substantially hurts performance: at $T_{\text{keep}} = 512$ accuracy drops from $48.4\%$ to $42.8\%$, and at $T_{\text{keep}} = 1024$ from $53.4\%$ to $49.2\%$. Beyond the overarching allocation paradigm, the specific mass-weighted segmentation and EMA credit are also vital. Removing mass-weighted quotas while keeping the rest of AMS intact reduces accuracy by about $2.4$ points at $T_{\text{keep}} = 512$ (from $48.4\%$ to $46.0\%$). Similarly, using fixed-length segments instead of mass-based adaptive segments yields a degradation, especially at larger budgets (from $53.4\%$ to $50.4\%$ at $T_{\text{keep}} = 1024$). Finally, disabling the EMA credit has little effect at $T_{\text{keep}} = 512$ but reduces accuracy by $2.6$ points at $T_{\text{keep}} = 1024$, indicating that history-aware allocation becomes more beneficial when the cache budget is larger and eviction decisions are less constrained.

We further test sensitivity to the compression interval, hidden-state (HS) buffer, and segmentation configuration. On DeepSeek-R1-Distill-Qwen-7B / MATH500, AMS-Expected remains robust across these settings (Appendix D): once the interval reaches 256 tokens, pass@1 changes by only a few points, and a wide range of segmentation hyperparameters yields nearly identical performance.

## 5 Conclusion

We introduced AMS, an adaptive mass-segmented framework for decoding-time KV compression. AMS reframes retention as allocate-then-score: attention-derived usage first forms adaptive temporal segments and assigns region-wise quotas, after which existing scorers select tokens locally. Across mathematical reasoning, cross-backbone transfer, larger model scales, and general long-context tasks,

AMS improves the accuracy–memory trade-off under constrained KV budgets. These results show that preserving regional cache structure is an effective complement to token-level importance scoring. We discuss limitations and potential societal impacts in Appendix A.

## 6 Acknowledgement

We thank EIT and IDT High Performance Computing Center for providing computational resources for this project. This work was supported by the 2035 Key Research and Development Program of Ningbo City under Grant No. 2025Z034.

## References

[1] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=RkRrPp7GKO.

[2] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles*, pages 611–626, 2023. doi: 10.1145/3600006.3613165. URL https://arxiv.org/abs/2309.06180.

[3] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2401.18079.

[4] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2201.11903.

[5] Zefan Cai, Wen Xiao, Hanshi Sun, Cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, Zhen Dong, Anima Anandkumar, Abedelkadir Asi, and Junjie Hu. R-KV: Redundancy-aware KV cache compression for reasoning models. *arXiv preprint arXiv:2505.24133*, 2025. doi: 10.48550/arXiv.2505.24133. URL https://arxiv.org/abs/2505.24133.

[6] Jiwon Song, Dongwon Jo, Yulhwa Kim, and Jae-Joon Kim. Reasoning path compression: Compressing generation trajectories for efficient LLM reasoning. In *Advances in Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=894Yo61h1P. NeurIPS 2025 Poster.

[7] Weian Mao, Xi Lin, Wei Huang, Yuxin Xie, Tianfu Fu, Bohan Zhuang, Song Han, and Yukang Chen. Tri-attention: Efficient long reasoning with trigonometric KV compression. *arXiv preprint arXiv:2604.04921*, 2026. doi: 10.48550/arXiv.2604.04921. URL https://arxiv.org/abs/2604.04921.

[8] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022. doi: 10.48550/arXiv.2205.14135. URL https://arxiv.org/abs/2205.14135.

[9] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.17453. ICLR 2024.

[10] Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state RNNs. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18724–18741, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1043. URL https://aclanthology.org/2024.emnlp-main.1043/.

[11] Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. Pyramidkv: Dynamic KV cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024. doi: 10.48550/arXiv.2406.02069. URL https://arxiv.org/abs/2406.02069.

[12] Jitai Hao, Yuke Zhu, Tian Wang, Jun Yu, Xin Xin, Bo Zheng, Zhaochun Ren, and Sheng Guo. Omnikv: Dynamic context selection for efficient long-context LLMs. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=ulCAPXYXfa. ICLR 2025.

[13] Dachuan Shi, Yonggan Fu, Xiangchi Yuan, Zhongzhi Yu, Haoran You, Sixu Li, Xin Dong, Jan Kautz, Pavlo Molchanov, and Yingyan Celine Lin. Lacache: Ladder-shaped KV caching for efficient long-context modeling of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 54892–54903. PMLR, 2025. URL https://proceedings.mlr.press/v267/shi25b.html.

[14] Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Ada-KV: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference. *arXiv preprint arXiv:2407.11550*, 2024. doi: 10.48550/arXiv.2407.11550. URL https://arxiv.org/abs/2407.11550.

[15] Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context LLM inference with retrieval and streaming heads. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/5c1ddd2e59df46fd2aa85c833b1b36ed-Paper-Conference.pdf. ICLR 2025.

[16] Yu Fu, Zefan Cai, Abedelkadir Asi, Wayne Xiong, Yue Dong, and Wen Xiao. Not all heads matter: A head-level KV cache compression method with integrated retrieval and reasoning. *arXiv preprint arXiv:2410.19258*, 2024. doi: 10.48550/arXiv.2410.19258. URL https://arxiv.org/abs/2410.19258.

[17] Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Danning Ke, Shikuan Hong, Yiwu Yao, and Gongyi Wang. Razorattention: Efficient kv cache compression through retrieval heads. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tkiZQlL04w.

[18] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: LLM knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*, 2024. doi: 10.48550/arXiv.2404.14469. URL https://arxiv.org/abs/2404.14469.

[19] Jinhan Chen, Jianchun Liu, Hongli Xu, Xianjun Gao, and Shilong Wang. Sablock: Semantic-aware KV cache eviction with adaptive compression block size. *arXiv preprint arXiv:2510.22556*, 2025. doi: 10.48550/arXiv.2510.22556. URL https://arxiv.org/abs/2510.22556.

[20] Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. Clusterkv: Manipulating LLM KV cache in semantic space for recallable compression. *arXiv preprint arXiv:2412.03213*, 2024. doi: 10.48550/arXiv.2412.03213. URL https://arxiv.org/abs/2412.03213.

[21] Zhiyuan Yu, Shijian Xiao, Zhangyue Yin, Xiaoran Liu, Lekai Xing, Wenzhong Li, Cam-Tu Nguyen, and Sanglu Lu. Protokv: Long-context knowledges are already well-organized before your query. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=kXhPkDaFbJ. ICLR 2026 Poster.

[22] Ziwei He, Jian Yuan, Haoli Bai, Jingwen Leng, and Bo Jiang. Treekv: Smooth key-value cache compression with tree structures. *arXiv preprint arXiv:2501.04987*, 2025. doi: 10.48550/arXiv.2501.04987. URL https://arxiv.org/abs/2501.04987.

[23] Zhiyuan Shi, Qibo Qiu, Feng Xue, Zhonglin Jiang, Li Yu, Jian Jiang, Xiaofei He, and Wenxiao Wang. Heterocache: A dynamic retrieval approach to heterogeneous KV cache compression for long-context LLM inference. *arXiv preprint arXiv:2601.13684*, 2026. doi: 10.48550/arXiv.2601.13684. URL https://arxiv.org/abs/2601.13684.

[24] Alessio Devoto, Maximilian Jeblick, and Simon Jégou. Expected attention: KV cache compression by estimating attention from future queries distribution. *arXiv preprint arXiv:2510.00636*, 2025. doi: 10.48550/arXiv.2510.00636. URL https://arxiv.org/abs/2510.00636.

[25] Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Yue Liu, Bo Li, Xuming Hu, and Xiaowen Chu. Chunkkv: Semantic-preserving KV cache compression for efficient long-context LLM inference. *arXiv preprint arXiv:2502.00299*, 2025. doi: 10.48550/arXiv.2502.00299. URL https://arxiv.org/abs/2502.00299. NeurIPS 2025.

[26] Yongqi An, Chang Lu, Kuan Zhu, Tao Yu, Chaoyang Zhao, Hong Wu, Ming Tang, and Jinqiao Wang. ReST-KV: Robust KV cache eviction with layer-wise output reconstruction and spatial-temporal smoothing. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=PhEHuo7oMm. ICLR 2026 Poster.

[27] Junyoung Park, Dalton Jones, Matt Morse, Raghavv Goel, Mingu Lee, and Chris Lott. Kediff: Key similarity-based KV cache eviction for long-context LLM inference in resource-constrained environments. *arXiv preprint arXiv:2504.15364*, 2025. doi: 10.48550/arXiv.2504.15364. URL https://arxiv.org/abs/2504.15364.

[28] Jialong Wu, Zhenglin Wang, Linhai Zhang, Yilong Lai, Yulan He, and Deyu Zhou. SCOPE: Optimizing key-value cache compression in long-context generation. *arXiv preprint arXiv:2412.13649*, 2024. doi: 10.48550/arXiv.2412.13649. URL https://arxiv.org/abs/2412.13649.

[29] Mengqi Liao, Lu Wang, Chaoyun Zhang, Zekai Shen, Xiaowei Mao, Si Qin, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Huaiyu Wan. G-KV: Decoding-time KV cache eviction with global attention. *arXiv preprint arXiv:2512.00504*, 2025. doi: 10.48550/arXiv.2512.00504. URL https://arxiv.org/abs/2512.00504.

[30] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. In *Advances in Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=JZfg6wGi6g.

[31] Isaac Rehg. KV-compress: Paged KV-cache compression with variable compression rates per attention head. *arXiv preprint arXiv:2410.00161*, 2024. doi: 10.48550/arXiv.2410.00161. URL https://arxiv.org/abs/2410.00161.

[32] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

[33] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygGQyrFvH. ICLR 2020.

[34] Yuan Feng, Haoyu Guo, Junlin Lv, S. Kevin Zhou, and Xike Xie. Taming the fragility of KV cache eviction in LLM inference. *arXiv preprint arXiv:2510.13334*, 2025. doi: 10.48550/arXiv.2510.13334. URL https://arxiv.org/abs/2510.13334.

[35] Junlong Tong, Zilong Wang, YuJie Ren, Peiran Yin, Hao Wu, Wei Zhang, and Xiaoyu Shen. From static inference to dynamic interaction: A survey of streaming large language models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 10237–10263. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.498. URL https://aclanthology.org/2026.findings-acl.498/.

[36] Yi Su, Zhenxu Tian, Dan Qiao, Yuechi Zhou, Juntao Li, and Min Zhang. Longflow: Efficient kv cache compression for reasoning models. *arXiv preprint arXiv:2603.11504*, 2026. doi: 10.48550/arXiv.2603.11504. URL https://arxiv.org/abs/2603.11504.

[37] Hui Zeng, Daming Zhao, Pengfei Yang, WenXuan Hou, Tianyang Zheng, Hui Li, Weiye Ji, and Jidong Zhai. Lethe: Layer- and time-adaptive kv cache pruning for reasoning-intensive llm serving. *arXiv preprint arXiv:2511.06029*, 2025. doi: 10.48550/arXiv.2511.06029. URL https://arxiv.org/abs/2511.06029.

[38] Akshat Ramachandran, Marina Neseem, Charbel Sakr, Rangharajan Venkatesan, Brucek Khailany, and Tushar Krishna. Thinkv: Thought-adaptive kv cache compression for efficient reasoning models. *arXiv preprint arXiv:2510.01290*, 2025. doi: 10.48550/arXiv.2510.01290. URL https://arxiv.org/abs/2510.01290.

[39] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. *arXiv preprint arXiv:2103.03874*, 2021. doi: 10.48550/arXiv.2103.03874. URL https://arxiv.org/abs/2103.03874.

[40] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/aca97732e30bcf1303bc22ac3924fd16-Abstract-Conference.html. ICLR 2024.

[41] TIGER-Lab. Aime25. Hugging Face dataset repository, 2025. URL https://huggingface.co/datasets/TIGER-Lab/AIME25. Accessed 2026-05-05.

[42] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. doi: 10.48550/arXiv.2110.14168. URL https://arxiv.org/abs/2110.14168.

[43] deepseek-ai. Deepseek-r1-distill-qwen-7b. Hugging Face model repository, 2025. URL https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-7B. Accessed 2026-05-05.

[44] DeepSeek-AI. Deepseek-r1-distill-qwen-32b. https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-32B, 2025. Distilled from DeepSeek-R1 and based on Qwen2.5-32B.

[45] open-thoughts. Openthinker3-7b. Hugging Face model repository, 2025. URL https://huggingface.co/open-thoughts/OpenThinker3-7B. Accessed 2026-05-05.

[46] Bohan Yu and Yekun Chai. EvolKV: Evolutionary KV cache compression for LLM inference. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 1673–1689. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-emnlp.88. URL https://aclanthology.org/2025.findings-emnlp.88/.

[47] Ngoc Bui, Hieu Trung Nguyen, Arman Cohan, and Rex Ying. Make each token count: Towards improving long-context performance with KV cache eviction, 2026. URL https://arxiv.org/abs/2605.09649.

[48] Haozhe Hu, Hao Wu, Anhao Zhao, Longwei Ding, Peiran Yin, Yunpu Ma, and Xiaoyu Shen. Beyond FLOPs: Benchmarking real inference acceleration of LLM pruning under a GEMM-centric taxonomy. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing*, pages 31283–31310. Association for Computational Linguistics, 2026. URL https://aclanthology.org/2026.emnlp-main.1371/.

# Appendix

## A Limitations and Impact Statement

**Limitations.** AMS is a training-free decoding-time allocation layer that uses attention-derived mass for adaptive segmentation. This design keeps the method lightweight and scorer-agnostic, but it also means that AMS relies on attention-derived usage as a proxy for long-range utility. Future work may explore task-aware or learned utility signals. Although our experiments cover multiple budgets, backbones, model scales, and task families, broader evaluation on additional architectures, multi-turn interactions, and production serving systems would further clarify deployment behavior. In addition, while AMS is compatible with gather-and-compact KV execution, its practical efficiency may still depend on the serving backend, batching strategy, and hardware implementation [48].

Our main mathematical reasoning experiments use greedy decoding with benchmark-specific generation limits. In particular, the 4096-token limit on MATH500 can affect answer completion and the interpretation of accuracy differences. AIME24 and AIME25 each contain only 30 problems, so individual accuracy estimates have substantial finite-sample uncertainty. Moreover, region-level coverage does not guarantee the retention of isolated critical tokens, which limits performance on exact-retrieval tasks such as NIAH. Although the reported diagnostics support an association between reduced Region Wipe-out and more stable generation, they do not establish a causal relationship. Finally, production-serving throughput under continuous batching, prefix caching, and optimized attention kernels remains to be evaluated.

**Impact Statement.** AMS reduces KV-memory cost for long-context inference, which may improve efficiency and accessibility of long-form language-model applications. At the same time, lowering inference cost can also make it easier to deploy long-form generation systems at scale, including systems that may generate misleading, harmful, or otherwise low-quality content. AMS should therefore be used with the same safety, monitoring, evaluation, and deployment safeguards as the underlying language model. The method does not introduce new training data or new model capabilities by itself, but it can improve the efficiency of existing models, so downstream risks mainly depend on the application context and the base model being compressed.

## B Full Experimental and Implementation Details

This appendix provides the full experimental protocol omitted from the main text. Appendix B.1 describes datasets and evaluation; Appendix B.2 summarizes baseline configurations and budget enforcement; Appendix B.4 gives AMS implementation details; Appendix D reports hyperparameters and sensitivity studies; and Appendix E discusses compatibility with vLLM-style paged KV serving.

### B.1 Datasets and Evaluation Protocol

**Benchmarks.** We evaluate decoding-time KV compression on four mathematical reasoning benchmarks:

- **MATH500** is a 500-problem subset of the MATH olympiad-style benchmark, containing competition-level problems across algebra, number theory, combinatorics, and geometry. Each instance is a free-form question with a short, typically numeric or symbolic, final answer.
- **AIME24** and **AIME25** consist of 30 problems each in the style of the American Invitational Mathematics Examination (AIME). Each problem admits a single integer answer in the range 0 to 999, and we follow the standard evaluation that counts a prediction as correct if the final integer matches the ground-truth answer.
- **GSM8K** is a collection of grade-school math word problems with natural-language solutions. We use the standard test split and report pass@1 accuracy following the official evaluation script.

For all these benchmarks, we treat the problem statement as the model input and evaluate accuracy on a single final answer per problem (pass@1).

**Data splits and subsampling.** Unless otherwise noted, all main results are reported on the full official test split of each benchmark: 500 problems for MATH500, 30 problems each for AIME24 and AIME25, and the standard test set for GSM8K. In some ablation studies (Appendix D), we additionally report results on a randomly sampled $10\%$ subset of MATH500 for efficiency; sampling is performed once with a fixed random seed and kept fixed across methods.

**Generality Benchmarks.** To rigorously assess the structural preservation capabilities of AMS beyond mathematical reasoning, we additionally incorporate broad long-context evaluations. These include LONGBENCH for general long-context comprehension, REPOBENCH-P for cross-file code completion, TRIVIAQA for open-domain question answering, and the NEEDLE-IN-A-HAYSTACK (NIAH) stress test for scattered structural retrieval. For these generality benchmarks, we adopt their respective standard evaluation scripts and metrics (e.g., accuracy, F1 score, or retrieval success rate) under the identical decoding-time KV constraints used for our primary math evaluations.

**Prompting and generation.** For each benchmark, the input consists of an optional context prefix (empty for mathematical reasoning tasks), followed by the problem text and an answer instruction. All main mathematical reasoning experiments use deterministic greedy (argmax) decoding, with exactly one rollout and one final output per problem. We set max_new_tokens to 4096 for MATH500 and GSM8K, and to 32768 for AIME24 and AIME25. Within each benchmark, Full KV, all baselines, and AMS variants use the same prompts, generation limits, decoding settings, and evaluation procedures. The fixed random seed is used for evaluation-pipeline and subset reproducibility. No problem-specific tuning is performed.

**Full-KV rollout lengths and compression strength.** We define the Full-KV rollout length as the total number of prompt and generated tokens.On MATH500, the mean Full-KV rollout length is 2,517.5 tokens, with a standard deviation of 1,231.6 and a median of 2,162.5 tokens. On MATH500, based on the per-example Full-KV rollout lengths, target KV budgets of 256, 512, and 1024 correspond to approximate mean retained ratios of 14.3%, 27.7%, and 51.1%, respectively, with mean compression factors of 9.83, 4.92, and 2.49. For AIME25, the mean Full-KV rollout length is 22757.6 tokens, with a median of 32850 tokens. At a target KV budget of 512, the approximate mean retained ratio is 6.09%.

**Answer extraction and scoring.** We follow the official evaluation scripts provided with each benchmark. For MATH500, AIME24, and AIME25, the scoring code extracts a single final answer from the model output (e.g., by parsing the last boxed or integer expression) and compares it to the ground-truth answer after simple normalization (such as stripping whitespace and ignoring formatting). A problem is counted as correct if this normalized prediction matches the reference answer, and we report the resulting pass@1 accuracy.

**Compression schedule.** All experiments use an uncompressed prefill stage, and KV-cache compression is applied only during decoding. For the main MATH500 and AIME experiments, recompression is normally triggered every $I = 512$ generated tokens, with per-layer target cache budgets $T_{\text{keep}} \in \{256, 512, 1024\}$.

For NIAH, an additional cache-length-based trigger is enabled to accommodate its long-input, short-output workload. The corresponding configuration is described in Appendix B.2. Within each benchmark, all compressed methods share the same compression controller, triggering conditions, and target budget.

### B.2 Baseline Configurations

In this section we summarize the KV-cache compression baselines used in the main experiments (Table 1) and describe the common decoding-time configuration under which they are evaluated.

**Decoding-time compression setting.** All baselines we compare to operate as decoding-time KV cache compression policies: the model first runs a standard prefill pass on the full prompt, and then periodically recompresses the KV cache during autoregressive decoding. Unless otherwise stated, we trigger a compression event every $I = 512$ generated tokens and cap the per-layer cache length at $T_{\text{keep}} \in \{256, 512, 1024\}$. For NIAH, we additionally

**Table 10:** Baseline methods and their exact `press_name` keys in the evaluator registry. All entries below refer to `DecodingPress(base_press=...)` wrappers unless stated otherwise. Here, $I$ and $T_{\text{keep}}$ denote the evaluation-time compression interval and target cache length, respectively; in all main experiments they are set by the unified evaluation controller rather than fixed by the registry defaults.

| Paper name | press_name (exact) | Registry construction (summary) |
|---|---|---|
| No-Compression | `no_press` | None (no KV compression) |
| StreamingLLM | `decoding_streaming_llm` | DecodingPress(base_press=StreamingLLMPress()) |
| TOVA | `decoding_TOVA` | DecodingPress(base_press=TOVAPress()) |
| TriAttention | `decoding_triattention` | DecodingPress(base_press=TriAttentionPress()) |
| KeyDiff | `decoding_keydiff` | DecodingPress(base_press=KeyDiffPress()) |
| PyramidKV | `decoding_pyramidkv` | DecodingPress(base_press=PyramidKVPress()) |
| AdaKV-ExpE2 | `decoding_adakv_expected_attention_e2` | DecodingPress(base_press=AdaKVPress(ExpectedAttentionPress(epsilon=1e-2))) |
| ChunkKV-Expected | `decoding_chunkkv_expected_attention` | DecodingPress(base_press=ChunkKVPress(press=ExpectedAttentionPress(epsilon=1e-2),chunk_length=20),compression_interval=I,target_size=T_keep,hidden_states_buffer_size=256) |
| R-KV | `decoding_rkv` | DecodingPress(base_press=RKVPress(window_size=8,kernel_size=7,mix_lambda=0.07,retain_ratio=0.1)) |
| RPC | `decoding_rpc` | DecodingPress(base_press=ReasoningPathPress(window_size=32,kernel_size=7),compression_interval=I,target_size=T_keep,hidden_states_buffer_size=64) |
| AMS-TOVA (ours) | `decoding_adaptivesegmenthead_TOVA` | DecodingPress(base_press=AdaptiveMassSegmentWrapperHeadPress(base_press=TOVAPress(...),...)) |
| AMS-Expected (ours) | `decoding_adaptive_segment_head_expected_attention` | DecodingPress(base_press=AdaptiveMassSegmentWrapperHeadPress(base_press=ExpectedAttentionPress(epsilon=1e-2),...)) |
| AMS-TriAttention (ours) | `decoding_adaptivesegmenthead_triattention` | DecodingPress(base_press=AdaptiveMassSegmentWrapperHeadPress(base_press=TriAttentionPress(),...)) |

enable a cache-length-based trigger with `trigger_on_k_len=True`, `target_size=512`, and `min_steps_before_k_len_trigger=8`. This allows compression to occur after decoding begins without waiting for 512 newly generated tokens. The same NIAH trigger configuration is used for all compared compressed methods. At each event, the compressor takes the current KV cache as input and returns a subset of positions to retain; the remaining entries are discarded from the cache. Across all methods, we always preserve a small number of fixed sink tokens at the beginning of the sequence and a suffix of the most recent tokens to stabilize generation.

**Unified evaluation protocol.** Within each benchmark, all compared compressed methods use the same effective KV budget and task-specific compression trigger settings. The main MATH500 and AIME experiments use target budgets in $\{256, 512, 1024\}$, while GSM8K uses budgets of 64 and 128 tokens. Gather-based methods (StreamingLLM, TOVA, TriAttention, KeyDiff, PyramidKV, and our AMS variants) physically compress the cache to $T_{\text{keep}}$, while AdaKV-ExpE2 is mask-based and enforces the budget only via attention masks.

**Notes on AdaKV-ExpE2.** In our experiments, AdaKV-ExpE2 denotes an AdaKV-style head-adaptive budget allocation instantiated with an expected-attention scorer (we use $\epsilon = 10^{-2}$).

As a result, all baselines and AMS variants are evaluated under identical per-layer KV budgets and compression schedules. The only difference between methods lies in how they use attention statistics or structural heuristics to decide which tokens to retain within the fixed budget.

**Implementation details for reproducibility.** For completeness, we briefly summarize how these baselines are exposed in our codebase. All decoding-time baselines are instantiated via a unified method name and executed under a decoding-only controller. This controller applies the same compression schedule ($I = 512$, $T_{\text{keep}} \in \{256, 512, 1024\}$) across methods and, for scoring-based approaches, uses a fixed HS buffer of 256 tokens. The no-compression setting is implemented by disabling the controller entirely, thereby keeping the full KV cache throughout decoding.

Beyond the methods listed in Tables 11 and 12, our framework also includes additional registry entries (e.g., Knorm, QFilter, and composite variants). We focus on the baselines above in the main paper as representative, strong training-free methods covering streaming-style eviction, attention-based scoring, head-adaptive allocation, and layer-adaptive allocation.

### B.3 Existing Assets and Licenses

We use publicly available datasets, backbone models, and baseline implementations only for research evaluation. Table 13 summarizes the main existing assets used in this paper. We cite the original sources in the main paper or appendix and follow the corresponding licenses or terms of use where publicly available. No new dataset or pretrained model is introduced in this work.

### B.4 Implementation Details and Pseudocode for AMS

AMS is implemented as a training-free wrapper around a base scoring rule (e.g., TOVA or an expected-attention scorer). At each compression event and for each layer and KV head, AMS (1) converts recent attention usage into a normalized quality-mass distribution along the sequence, (2) partitions the sequence into adaptive segments according to this mass, (3) allocates a segment-wise quota of kept tokens under a global budget, and (4) applies the base scores within each segment to select tokens. An optional exponential moving average (EMA) credit makes the mass history-aware across compression events.

**Default configuration.** Unless otherwise stated, we use a single AMS configuration across all datasets and models in the main experiments:

- *Mass construction.* Quality mass is derived from recent-window attention statistics over the last $W$ decoding queries (using the same $W$ for all layers). Specifically, to reduce local noise, we apply a small 1D average-pooling kernel to the sequence of usage scores. Furthermore, for recent suffix positions that have fewer valid attending queries due to causal masking, we pad the missing observations with the maximum score observed in the window to avoid underestimating newly generated tokens. The scores are then normalized into a mass distribution. The EMA credit uses a decay parameter $\lambda \in (0, 1)$ and mixing coefficient $\beta \in [0, 1]$ as described in Section 3 of the main text.
- *Segmentation.* For each head, we construct adaptive segments by thresholding the cumulative mass curve with a target segment mass $\Delta$, then apply simple split and merge heuristics to keep segment lengths within $[L_{\min}, L_{\max}]$. The same $(\Delta, L_{\min}, L_{\max})$ are used for all layers.
- *Quota allocation.* Each segment receives a minimum quota $q_{\min}$ and an additional quota proportional to its mass under the global budget $T_{\text{keep}}$. We always keep a small number of sink tokens at the beginning of the sequence and a suffix of the most recent tokens; these must-keep positions are included on top of the segment-wise selection and take precedence when enforcing the overall budget.
- *In-segment selection.* Given segment quotas, we apply the base scorer (TOVA or expected attention) independently within each segment and select the top-scoring positions up to the quota. The union of in-segment selections and must-keep positions is then trimmed or backfilled to match $T_{\text{keep}}$ before gathering the compressed cache.

In all main results, these hyperparameters are kept fixed across datasets and backbones. The exact hyperparameter values used in the main experiments are summarized in Table 15, and Appendix D reports additional sensitivity studies.

**Attention usage and GQA handling.** Let $T$ denote the current KV-cache length and $W$ the number of queries in the recent observation window. For each KV head, attention weights are aligned with the current cache positions and aggregated over the most recent $W$ queries. We then apply one-dimensional average pooling along the cache-position dimension to reduce local fluctuations.

Under grouped-query attention (GQA), the attention scores of all query heads associated with the same KV head are averaged to obtain one usage vector per KV head. This usage vector is subsequently

normalized into the quality-mass distribution defined in Eq. (1). Segmentation, quota allocation, and token selection are performed independently for each KV head. All query heads mapped to the same KV head therefore share the same retained K/V entries.

All position indices in Sections 3.2 and 3.3 refer to current KV-cache positions rather than absolute generation positions. The $k$-th cumulative-mass threshold defines an initial cut position $b_k$, while the $i$-th final segment is denoted by $[a_i, b_i)$, where $a_i$ and $b_i$ are the inclusive left and exclusive right boundaries, respectively. After compression, the retained entries preserve their chronological order, although adjacent cache positions may correspond to non-consecutive original generation positions.

**Pseudocode.** Algorithm 1 summarizes one AMS compression event for a single layer and KV head. The actual implementation vectorizes these operations over batch and heads, but the logic is identical.

## C Additional Mechanistic Diagnostics

**Mitigating middle-context loss and Region Wipe-out.** Figure 4(a) shows that token-wise selection suffers from a strong "lost in the middle" pattern: TOVA under-retains tokens in the middle portion of the reasoning context, where intermediate derivations often appear. AMS counteracts this effect through segment-wise quotas, improving retention in the middle region and reducing the Region Wipe-out Rate from $11.3\%$ to $8.7\%$.

We further measure temporal stability using the retained-set IoU between consecutive recompression events. AMS consistently improves retained-set IoU across mathematical domains and transformer depth stages, indicating that it preserves the reasoning skeleton more stably over time. Detailed layer-wise and task-wise diagnostics, including the full IoU table and stability visualizations, are provided in Table 14.

**Suppressing Repetition Collapse.** Figure 4(b) shows that repetition collapse becomes more severe as problem difficulty increases. Harder problems require longer reasoning trajectories and therefore trigger more compression events, making token-wise eviction more likely to fragment the cache. Under this stress, TOVA exhibits a substantially higher N-gram repetition rate, reaching $25.9\%$ on Level-5 problems, while AMS reduces it to $19.4\%$. This suggests that region-level coverage helps the model avoid local reasoning loops and converge more reliably.

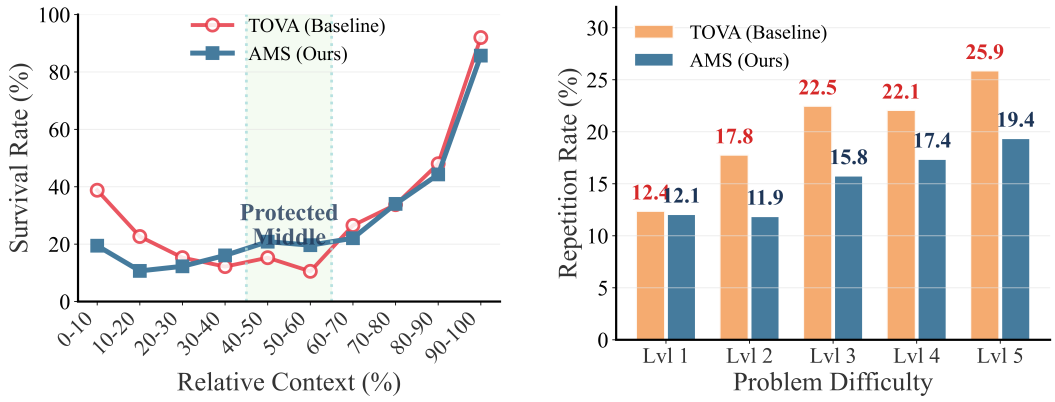
In the subset where AMS succeeds but TOVA fails, TOVA's outputs show a much higher repetition rate and tend to continue for more tokens without reaching a stable conclusion. In contrast, AMS produces shorter and more stable reasoning trajectories, supporting the view that region-wise allocation improves long-context reasoning by preventing compression-induced state fragmentation.

**Event-matched Region Wipe-out analysis.** To examine whether reduced Region Wipe-out can be explained solely by shorter generations and fewer compression events, we further analyze the complete MATH500 compression traces at $T_{\text{keep}} = 256$.

Among 228 paired examples where TOVA and AMS-TOVA both produce incorrect answers and undergo exactly the same number of compression events, AMS-TOVA reduces the mean trace-based wipe-out severity by 0.193.

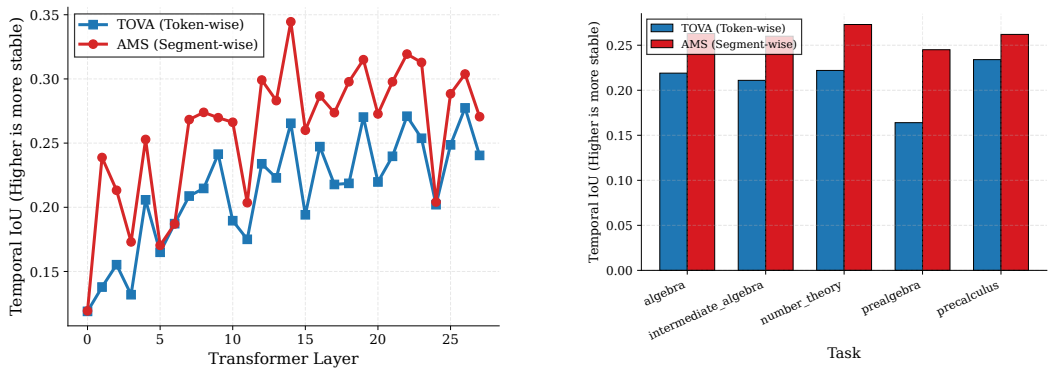
This comparison controls for problem identity, correctness outcome, and compression-event count. It shows that AMS induces different contiguous-region deletion patterns even when compression exposure is matched.

We provide additional diagnostics supporting the mechanistic analysis. In particular, we measure the temporal stability of retained KV states using retained-set IoU between consecutive recompression events, and report results across both mathematical subdomains and transformer depth stages.




**(a)** Spatial retention rate **(b)** Repetition collapse

**Figure 4: Mechanistic insights on MATH500.** (a) TOVA under-retains the middle portion of the reasoning context, while AMS improves middle-context coverage through segment-wise quotas. (b) Repetition collapse increases with problem difficulty under token-wise eviction; AMS suppresses this degradation.




**(a)** Layer-wise stability **(b)** Task-wise stability

**Figure 5: Temporal stability of retained context.** AMS consistently achieves higher temporal retained-set IoU than TOVA across transformer layers and mathematical sub-tasks. For the token-wise TOVA baseline, consecutive retained tokens are grouped as proxy segments for direct comparison.

**Table 11:** Decoding-time KV-cache compression baselines and their qualitative behavior. All methods are training-free and share the same cache budget $T_{\mathrm{keep}}$ and compression interval $I$ in our experiments.

| Method | Type | Description |
|---|---|---|
| No-Compression | reference | Standard decoding without any KV compression; serves as a no-compression reference. |
| StreamingLLM | streaming-style eviction | Maintains a small set of globally important positions together with a sliding window of recent tokens, designed for stable long-context streaming generation. |
| TOVA | attention-based pruning | Scores cached tokens using attention weights from the most recent query token and keeps the highest-scoring positions under the global budget. |
| TriAttention | pre-RoPE geometric pruning | Estimates key importance by leveraging intrinsic pre-RoPE Q/K concentration to score keys according to their spatial distance preferences. |
| KeyDiff | gradient-free importance | Uses differences between consecutive key vectors as an importance signal and retains tokens whose keys change most significantly. |
| PyramidKV | layer-adaptive budgeting | Allocates larger effective budgets to lower layers and smaller budgets to higher layers under a global cache limit in its original formulation; in our experiments it is evaluated under the same per-layer cap $T_{\mathrm{keep}}$ as other methods. |
| AdaKV-ExpE2 | head-adaptive, masking-based | Allocates retention budgets adaptively across attention heads using an expected-attention scoring rule (with a small regularization coefficient); in our implementation, eviction is implemented via masking rather than physically removing KV entries. |
| ChunkKV-Expected | fixed-segment, scorer-wrapped | Wraps an underlying scorer with chunk-wise token selection: it first computes global token scores, then ranks chunks by aggregated score and keeps tokens chunk by chunk to preserve local semantic coherence under compression. |
| R-KV | reasoning-specific scoring | Uses a redundancy-aware score that combines recent attention importance with a cosine-similarity-based redundancy penalty over historical keys, while protecting the prompt prefix and most recent tokens. |
| RPC | reasoning-specific scoring | Uses a reasoning-path style score computed from a window of recent queries, aggregates attention over the recent path, smooths scores with 1D average pooling, and protects the prompt prefix and recent tokens. |
| AMS-TOVA (ours) | segment-wise allocation | Wraps a TOVA-style scorer with our adaptive mass-segmented allocation: attention-derived quality mass is used to form segments and assign region-wise quotas, after which TOVA scores are applied within each segment. |
| AMS-Expected (ours) | segment-wise allocation | Same as AMS-TOVA but using an expected-attention-based scorer instead of TOVA. |
| AMS-TriAttention (ours) | segment-wise allocation | Wraps the TriAttention geometric scorer with adaptive mass-segmented allocation. |

| Method (paper) | Code identifier | Stage | KV handling |
|---|---|---|---|
| No-Compression | `no_press` | prefill & decode | full KV (No Compression) |
| StreamingLLM | `decoding_streaming_llm` | decode-only | gather-based |
| TOVA | `decoding_TOVA` | decode-only | gather-based |
| TriAttention | `decoding_triattention` | decode-only | gather-based |
| KeyDiff | `decoding_keydiff` | decode-only | gather-based |
| PyramidKV | `decoding_pyramidkv` | decode-only | gather-based |
| AdaKV-ExpE2 | `decoding_adakv_expected_attention_e2` | decode-only | **mask-based (no gather)** |
| ChunkKV-Expected | `decoding_chunkkv_expected_attention` | decode-only | gather-based |
| R-KV | `decoding_rkv` | decode-only | gather-based |
| RPC | `decoding_rpc` | decode-only | gather-based |

**Table 12:** Main baselines and the corresponding identifiers used in our open-source implementation. "decode-only" indicates that the compressor is applied only during autoregressive generation, not during prefill.

| Asset | Type | Usage in this paper | License / terms |
|---|---|---|---|
| MATH500 | Dataset | Mathematical reasoning evaluation | Public research benchmark; original source cited |
| AIME24/AIME25 | Dataset | Competition-style mathematical reasoning evaluation | Public benchmark; original source cited |
| GSM8K | Dataset | Grade-school math word-problem evaluation | Public research benchmark; original source cited |
| LONGBENCH | Dataset | Long-context generalization evaluation | Public research benchmark; original source cited |
| REPOBENCH-P | Dataset | Code-completion evaluation | Public research benchmark; original source cited |
| TRIVIAQA | Dataset | Open-domain QA evaluation | Public research benchmark; original source cited |
| NEEDLE-IN-A-HAYSTACK | Evaluation protocol | Sparse retrieval stress test | Public evaluation protocol; original source cited |
| DEEPSEEK-R1-DISTILL-QWEN-7B/32B | Backbone model | Main and scalability experiments | Public model; original license or model card terms followed |
| OPENTHINKER3-7B | Backbone model | Cross-backbone generalization experiments | Public model; original license or model card terms followed |
| KVPRESS | Software framework | Decoding-time KV-compression implementation | Public software package; original license followed |
| Baseline KV compressors | Algorithms / implementations | Comparative evaluation | Original papers and available implementations cited |

**Table 13:** Existing assets used in this paper.

**Algorithm 1** Adaptive Mass-Segmented KV Compression (single layer/head)

**Require:** KV cache $K, V \in \mathbb{R}^{B \times H_{kv} \times T \times D}$
**Require:** global keep budget $T_{\text{keep}}$
**Require:** recent-window usage scores for the last $W$ decoding queries
**Require:** base scorer BaseScore returning $g \in \mathbb{R}^{B \times H_{kv} \times T}$
**Require:** hyperparameters: target segment mass $\Delta$, minimum quota $q_{\min}$, segment length bounds $L_{\min}, L_{\max}$, must-keep sizes $n_{\text{sink}}, n_{\text{last}}$, EMA parameters $\lambda, \beta$
**Ensure:** compressed cache $K', V'$ with length $T_{\text{keep}}$

**Step 1: quality mass from recent attention**
aggregate usage of each cached position from the last $W$ queries to obtain scores $u_1, \ldots, u_T$
optionally smooth $\{u_t\}$ along the sequence (e.g., short 1D averaging)
normalize to a non-negative mass distribution $m^{(\text{cur})}$ via $m_t^{(\text{cur})} \propto \max(u_t, 0) + \varepsilon$, $\sum_t m_t^{(\text{cur})} = 1$
**Step 2: history-aware mass (EMA credit)**
**if** credit is used **then**
  update credit $c \leftarrow \lambda c + (1 - \lambda)\, m^{(\text{cur})}$
  $\tilde{c} \leftarrow \text{normalize}(c)$
  $m \leftarrow \text{normalize}\big(\beta\, m^{(\text{cur})} + (1 - \beta)\, \tilde{c}\big)$
**else**
  $m \leftarrow m^{(\text{cur})}$
**end if**
**Step 3: mass-based adaptive segmentation**
compute prefix sums $c(t) = \sum_{u=1}^{t} m_u$ for $t = 1, \ldots, T$
define thresholds $k\Delta$ for $k = 1, 2, \ldots$ and find cut points $b_k = \min\{t : c(t) \geq k\Delta\}$
form initial segments from $\{0, \{b_k\}, T\}$ as intervals $[a_i, b_i)$
**for** each segment $[a_i, b_i)$ **do**
  **if** $b_i - a_i > L_{\max}$ **then**
    split into roughly equal subsegments
  **else if** $b_i - a_i < L_{\min}$ **then**
    merge with a neighboring segment
  **end if**
**end for**
**Step 4: segment-wise quota allocation**
for each segment $i$, compute length $L_i = b_i - a_i$ and mass $M_i = \sum_{t=a_i}^{b_i - 1} m_t$
assign minimum quota $q_i^{\min} = \min(q_{\min}, L_i)$
$T_{\text{rem}} \leftarrow T_{\text{keep}} - \sum_i q_i^{\min}$
distribute $T_{\text{rem}}$ proportionally to $\{M_i\}$: $q_i \leftarrow q_i^{\min} + \text{round}\big(T_{\text{rem}} M_i / \sum_j M_j\big)$
clip $q_i \leq L_i$ and adjust $\{q_i\}$ so that $\sum_i q_i = T_{\text{keep}}$
**Step 5: must-keep set and in-segment selection**
define must-keep positions $\mathcal{I}_{\text{must}}$ as the first $n_{\text{sink}}$ tokens and the last $n_{\text{last}}$ tokens (when feasible)
obtain token scores $g = \text{BaseScore}(K, V)$
**for** each batch index $b$ and KV head $h$ **do**
  **for** each segment $[a_i, b_i)$ with quota $q_i$ **do**
    select top-$q_i$ indices in $\{a_i, \ldots, b_i - 1\}$ using $g_{b,h,:}$, forming $\mathcal{I}_{\text{seg},i}^{(b,h)}$
  **end for**
  $\mathcal{I}_{\text{seg}}^{(b,h)} \leftarrow \bigcup_i \mathcal{I}_{\text{seg},i}^{(b,h)}$
  $\mathcal{I}^{(b,h)} \leftarrow \mathcal{I}_{\text{seg}}^{(b,h)} \cup \mathcal{I}_{\text{must}}$
  remove duplicates from $\mathcal{I}^{(b,h)}$ and sort
  trim lowest-scoring non-must-keep indices if $|\mathcal{I}^{(b,h)}| > T_{\text{keep}}$
  backfill highest-scoring remaining indices if $|\mathcal{I}^{(b,h)}| < T_{\text{keep}}$
**end for**
**Step 6: gather compressed cache**
gather $K', V'$ along the sequence axis using $\mathcal{I}^{(b,h)}$ for all $(b, h)$
**return** $K', V'$

**Table 14: Additional temporal-stability diagnostics.** AMS improves retained-set IoU across mathematical domains and transformer depth stages while maintaining interpretable adaptive segments.

| Subset / Stage | Base IoU | AMS IoU | $\Delta$ IoU | Avg #Segments | Avg Seg. Len |
|---|---|---|---|---|---|
| *Across Mathematical Domains* | | | | | |
| Algebra | 0.219 | 0.260 | **+0.042** | 8.05 | 138.8 |
| Intermediate Alg. | 0.211 | 0.258 | **+0.047** | 8.80 | 138.1 |
| Number Theory | 0.221 | 0.273 | **+0.052** | 7.61 | 135.6 |
| Prealgebra | 0.164 | 0.244 | **+0.080** | 7.63 | 142.1 |
| Precalculus | 0.234 | 0.262 | **+0.028** | 8.36 | 145.2 |
| *Across Transformer Depth* | | | | | |
| Early (Layers 0 to 7) | 0.164 | 0.203 | **+0.039** | 10.01 | 142.4 |
| Middle (Layers 8 to 19) | 0.224 | 0.281 | **+0.057** | 7.73 | 131.5 |
| Late (Layers 20 to 27) | 0.244 | 0.284 | **+0.040** | 7.18 | 147.8 |

## D Additional Ablations

**Overview.** Unless otherwise stated, we use the default AMS configuration in Table 15. We additionally provide a sensitivity study on compression interval and HS buffer in Table 17. Here `metric_main` is the raw metric reported by our evaluation script (see caption for details).

| Parameter | Value |
|---|---|
| Compression interval $I$ | 512 |
| HS buffer | 256 |
| Segment mass $\Delta$ | 0.1 |
| Min / max segment length | 16 / 256 |
| Min quota per segment $q_{\min}$ | 1 |
| EMA decay $\lambda$ | 0.9 |
| Mass mixing $\beta$ | 0.9 |
| Recent window size $W$ | 128 |
| Number of sink tokens | 4 |

**Table 15:** Final AMS hyperparameter configuration used for all experiments unless otherwise stated.

**Sensitivity to recent-window size $W$.** Since AMS relies on recent attention usage to compute the quality-mass distribution, we verified its sensitivity to the window size $W$. As shown in Table 16, on a joint sensitivity sweep over the MATH500 subset ($T_{\text{keep}} = 512$), AMS-Expected attained a perfectly stable pass@1 of 50.0% for $W \in \{16, 32, 64\}$. This indicates that the EMA temporal smoothing effectively filters out instantaneous attention jitter, making the segmentation logic highly robust to the exact choice of the local attention window size.

**Table 16: Sensitivity to Recent-Window Size.** Ablation of the window size $W$ used for mass estimation in AMS-Expected on MATH500. Performance remains perfectly stable.

| **Window Size ($W$)** | **16** | **32** | **64** |
|---|---|---|---|
| Pass@1 (%) | 50.0 | 50.0 | 50.0 |

**Supplementary AIME25 ablations.** We additionally evaluate the contributions of AMS components on AIME25 using a target KV budget of 512 and two matched sampling seeds (42 and 43), yielding 60 problem-seed outcomes per variant. Full AMS-TOVA achieves 8/60 correct outcomes (13.33%), compared with 2/60 (3.33%) using global head-shared allocation, 4/60 (6.67%) using fixed-length segmentation, and 5/60 (8.33%) without EMA credit.

## E Compatibility with vLLM-Style Paged KV Serving

**Motivation.** Our main experiments are implemented through the HuggingFace/KVPress execution path, where KV compression is naturally expressed as a gather operation over dense per-layer KV tensors. In contrast, production serving systems such as vLLM store the KV cache in paged physical blocks and use block tables to map logical sequence positions to physical cache slots [2]. This paged layout is highly effective for batched serving and memory management, but it raises a systems question for AMS: our retention policy may produce head-specific keep sets $\mathcal{I}_{b,h}$, while a standard paged block table maps each logical token position to a physical slot shared across KV heads.

**Policy-layout separation.** AMS naturally separates the retention policy from the physical cache layout. The policy component decides which original token positions each KV head should retain, producing keep indices $\mathcal{I} \in \mathbb{N}^{B \times H_{kv} \times T_{\text{keep}}}$. The serving runtime is then responsible for deciding where the corresponding KV vectors are stored in physical memory. This separation means that AMS does not require changing its segmentation, quota allocation, or in-segment scoring algorithm in order to target a paged-KV backend. Instead, the runtime only needs to materialize the selected KV entries into a compact paged layout.

| $T_{\text{keep}}$ | Interval | HS buffer | #Samples | Correct | Pass@1 (%) | sec/sample | peak GB |
|---|---|---|---|---|---|---|---|
| 512 | 128 | 0 | 50 | 20 | 40.0 | 62.3 | 14.35 |
| | 128 | 64 | 50 | 20 | 40.0 | 63.9 | 14.35 |
| | 128 | 256 | 50 | 20 | 40.0 | 61.9 | 14.35 |
| | 256 | 0 | 50 | 24 | 48.0 | 57.0 | 14.38 |
| | 256 | 64 | 50 | 20 | 40.0 | 61.4 | 14.35 |
| | 256 | 256 | 50 | 24 | 48.0 | 58.9 | 14.38 |
| | 512 | 0 | 50 | 25 | 50.0 | 60.2 | 14.40 |
| | 512 | 64 | 50 | 24 | 48.0 | 62.1 | 14.35 |
| | 512 | 256 | 50 | 25 | 50.0 | 57.1 | 14.40 |
| | 1024 | 0 | 50 | 24 | 48.0 | 57.2 | 14.42 |
| | 1024 | 64 | 50 | 24 | 48.0 | 55.2 | 14.38 |
| | 1024 | 256 | 50 | 24 | 48.0 | 59.6 | 14.42 |
| 1024 | 128 | 0 | 50 | 24 | 48.0 | 56.5 | 14.39 |
| | 128 | 64 | 50 | 25 | 50.0 | 56.6 | 14.38 |
| | 128 | 256 | 50 | 24 | 48.0 | 59.0 | 14.39 |
| | 256 | 0 | 50 | 25 | 50.0 | 55.4 | 14.42 |
| | 256 | 64 | 50 | 24 | 48.0 | 56.7 | 14.38 |
| | 256 | 256 | 50 | 25 | 50.0 | 55.0 | 14.42 |
| | 512 | 0 | 50 | 26 | 52.0 | 55.4 | 14.42 |
| | 512 | 64 | 50 | 25 | 50.0 | 52.9 | 14.38 |
| | 512 | 256 | 50 | 26 | 52.0 | 55.3 | 14.42 |
| | 1024 | 0 | 50 | 26 | 52.0 | 54.2 | 14.45 |
| | 1024 | 64 | 50 | 26 | 52.0 | 54.9 | 14.41 |
| | 1024 | 256 | 50 | 26 | 52.0 | 53.3 | 14.45 |
| 2048 | 128 | 0 | 50 | 25 | 50.0 | 54.4 | 14.43 |
| | 128 | 64 | 50 | 25 | 50.0 | 54.6 | 14.42 |
| | 128 | 256 | 50 | 25 | 50.0 | 55.1 | 14.43 |
| | 256 | 0 | 50 | 25 | 50.0 | 54.8 | 14.47 |
| | 256 | 64 | 50 | 24 | 48.0 | 56.8 | 14.43 |
| | 256 | 256 | 50 | 25 | 50.0 | 55.9 | 14.47 |
| | 512 | 0 | 50 | 25 | 50.0 | 55.4 | 14.48 |
| | 512 | 64 | 50 | 25 | 50.0 | 54.9 | 14.44 |
| | 512 | 256 | 50 | 25 | 50.0 | 55.4 | 14.48 |
| | 1024 | 0 | 50 | 23 | 46.0 | 56.8 | 14.51 |
| | 1024 | 64 | 50 | 25 | 50.0 | 54.6 | 14.47 |
| | 1024 | 256 | 50 | 23 | 46.0 | 58.3 | 14.51 |

**Table 17:** Sweep of compression interval and HS buffer for AMS-Expected on MATH500 (DeepSeek-R1-Distill-Qwen-7B, fraction=0.1). Correct = metric_main in the csv.

**Head-wise compaction.** A simple but lossy vLLM adaptation would force AMS to keep whole blocks or to share one keep set across all KV heads. Although this would fit the existing block-table abstraction, it would change the AMS policy and weaken the head-wise region guarantees studied in this paper. We instead use a compression-time head-wise compaction view.

Let $\pi_{\text{old}}(b, p)$ denote the physical slot corresponding to the old logical position $p$ of request $b$, and let $\pi_{\text{new}}(b, t)$ denote the physical slot of compact position $t$ after allocating replacement blocks. For each layer, AMS compaction performs the conceptual copy

$$\begin{aligned} s^{\text{src}}_{b,h,t} &= \pi_{\text{old}}\big(b, \mathcal{I}_{b,h,t}\big), \\ s^{\text{dst}}_{b,t} &= \pi_{\text{new}}(b, t). \end{aligned} \tag{2}$$

| $W$ | $\Delta$ | $L_{\min}$ | $L_{\max}$ | $q_{\min}$ | keep_last | $n_{\text{sink}}$ | Pass@1 (%) | sec/sample | peak GB |
|---|---|---|---|---|---|---|---|---|---|
| 16 | 0.005 | 32 | 1024 | 32 | 16 | 4 | 50.0 | 55.9 | 14.48 |
| 16 | 0.005 | 64 | 1024 | 8 | 32 | 8 | 50.0 | 54.4 | 14.48 |
| 16 | 0.005 | 128 | 512 | 0 | 16 | 0 | 50.0 | 54.6 | 14.48 |
| 16 | 0.010 | 16 | 4096 | 16 | 128 | 8 | 50.0 | 58.6 | 14.48 |
| 16 | 0.010 | 32 | 1024 | 32 | 128 | 4 | 50.0 | 55.9 | 14.48 |
| 16 | 0.010 | 64 | 2048 | 16 | 16 | 0 | 50.0 | 56.4 | 14.48 |
| 16 | 0.010 | 128 | 1024 | 1 | 32 | 4 | 50.0 | 55.7 | 14.48 |
| 16 | 0.020 | 16 | 512 | 32 | 16 | 4 | 50.0 | 54.9 | 14.48 |
| 16 | 0.020 | 32 | 1024 | 8 | 64 | 8 | 50.0 | 55.1 | 14.48 |
| 16 | 0.020 | 64 | 2048 | 16 | 32 | 0 | 50.0 | 55.0 | 14.48 |
| 32 | 0.005 | 16 | 1024 | 32 | 16 | 4 | 50.0 | 56.2 | 14.48 |
| 32 | 0.005 | 32 | 4096 | 8 | 128 | 8 | 50.0 | 54.0 | 14.48 |
| 32 | 0.005 | 64 | 1024 | 0 | 16 | 0 | 50.0 | 54.8 | 14.48 |
| 32 | 0.010 | 16 | 2048 | 16 | 32 | 8 | 50.0 | 56.0 | 14.48 |
| 32 | 0.010 | 32 | 1024 | 32 | 64 | 4 | 50.0 | 54.6 | 14.48 |
| 32 | 0.010 | 64 | 4096 | 16 | 16 | 0 | 50.0 | 54.4 | 14.48 |
| 32 | 0.010 | 128 | 512 | 1 | 128 | 4 | 50.0 | 55.9 | 14.48 |
| 32 | 0.020 | 16 | 1024 | 32 | 16 | 4 | 50.0 | 56.5 | 14.48 |
| 32 | 0.020 | 32 | 512 | 8 | 64 | 8 | 50.0 | 54.9 | 14.48 |
| 32 | 0.020 | 64 | 2048 | 16 | 32 | 0 | 50.0 | 55.5 | 14.48 |
| 64 | 0.005 | 16 | 1024 | 32 | 16 | 4 | 50.0 | 56.5 | 14.48 |
| 64 | 0.005 | 32 | 4096 | 16 | 64 | 0 | 50.0 | 54.2 | 14.48 |
| 64 | 0.010 | 16 | 2048 | 8 | 32 | 8 | 50.0 | 55.7 | 14.48 |
| 64 | 0.010 | 32 | 1024 | 16 | 128 | 4 | 50.0 | 54.4 | 14.48 |
| 64 | 0.020 | 16 | 512 | 32 | 16 | 0 | 50.0 | 55.9 | 14.48 |

**Table 18:** Sensitivity sweep for segmentation-related hyperparameters in AMS-Expected on MATH500 (fraction=0.1). Pass@1 is computed from metric_main / num_samples in the csv.

The KV entries are then materialized as

$$\begin{aligned} K^{\text{new}}\left[s_{b,t}^{\text{dst}}, h, :\right] &\leftarrow K^{\text{old}}\left[s_{b,h,t}^{\text{src}}, h, :\right], \\ V^{\text{new}}\left[s_{b,t}^{\text{dst}}, h, :\right] &\leftarrow V^{\text{old}}\left[s_{b,h,t}^{\text{src}}, h, :\right]. \end{aligned} \tag{3}$$

After this copy, the serving engine can continue using an ordinary compact block table and the standard attention read path. The head-wise selection has already been materialized into the KV contents, so the steady-state attention kernel does not need an additional per-head indirection table. For decoder-only attention, this is equivalent to the dense gather formulation used in our experiments, because each KV head is attended independently and all retained entries are past tokens. For RoPE-based models, positional information is already encoded in the cached keys before compaction; the runtime must still keep the next generated token at the original logical sequence position rather than treating the compact KV length as the new autoregressive position.

**Paged-KV runtime compaction.** An end-to-end vLLM implementation would add a request-level compaction routine to the KV-cache manager. At a compression event, the serving runtime would:

1. materialize the current per-request KV view needed by the AMS selector;
2. call the AMS/KVPress selector to obtain head-wise keep indices $\mathcal{I} \in \mathbb{N}^{B \times H_{kv} \times T_{\text{keep}}}$;
3. allocate compact replacement blocks from the paged KV block pool;
4. launch a layout-aware GPU copy kernel that performs the per-head KV movement above for every attention layer;
5. replace the request's block-table row with the compact block IDs and free the old blocks after the copy completes; and
6. maintain separate bookkeeping for the logical decoding position and the compact physical KV length.

The last item is important because the compact cache length becomes $T_{\text{keep}}$, while the next generated token should still follow the original autoregressive position.

**Current implementation status.** The supplementary code implements this policy–layout contract in two levels. First, a standalone reference adapter constructs head-wise source and destination slot mappings and verifies that attention over the compacted cache matches dense attention over AMS-gathered KV tensors. Second, we provide a vLLM-style runtime path that connects AMS selection to paged-KV compaction through runtime hooks: AMS produces head-wise keep indices, the runtime compacts the physical KV blocks, updates the request block table, and reclaims old blocks before standard vLLM attention continues over the compact cache.

The runtime path supports decoupled mass and selection scores, including TOVA-style and expected-attention selection. This implementation is not an upstream vLLM patch; it is a runtime integration used to validate that AMS can preserve head-wise selections in a paged-KV serving layout without introducing per-head block tables in steady-state attention. We therefore report the main benchmark results using the unified decoding-time KV-compression controller, and use this vLLM path as an implementation and compatibility validation rather than as a fully optimized production-serving benchmark.

# F Hardware and Runtime Environment

All experiments were conducted on Linux servers equipped with NVIDIA A800 GPUs. Each GPU has 80 GB memory, and each evaluation job used 2 GPU(s) unless otherwise noted.

Our implementation is based on PyTorch 2.3.1 with CUDA 12.1 and the KVPRESS framework (v0.4.0) for decoding-time KV-cache compression. We use the HuggingFace TRANSFORMERS library (v4.57.3) together with ACCELERATE (v1.0.1) for model loading and inference, and standard scientific Python libraries including NumPy 2.3.5, SciPy 1.16.3, and pandas 2.3.3. Benchmark data are loaded via the DATASETS library (v2.21.0), and plots are generated using MATPLOTLIB (v3.10.7). Unless otherwise noted, all reported results are obtained under this runtime environment using the unified decoding-time KV-compression controller described in Appendix B.2. Our vLLM compatibility validation was conducted in an isolated CUDA-12.4 environment using a vLLM V1 runtime, without changing the main experimental environment.

## G Qualitative Case Studies on Failure Modes

To concretely illustrate the failure modes discussed in Section 1 and Section C, we provide qualitative examples from the MATH500 benchmark, comparing the generation trajectories of the baseline TOVA and our proposed AMS framework under the same KV budget ($T_{\text{keep}} = 512$).

### G.1 Case 1: Problem Drifting and Repetition Collapse

When the context budget is strictly token-wise, the model often drops the original problem constraints during long derivations. As shown in Table 19, TOVA initially attempts the polynomial division but eventually suffers from *Region Wipe-out*. Consequently, it forgets the original constant ($-3$), hallucinates a new problem (changing the coefficient to $-1$), and falls into a severe *repetition collapse*, indefinitely looping over the wrong equation. In contrast, AMS safely preserves the structural anchors, correctly identifying the coefficients and solving the problem steadily.

**Table 19: Case Study 1: Problem Drifting (Polynomial Division).** The baseline forgets the initial constants and drifts into a repetitive hallucination, while AMS maintains state stability.

**Question:** Find the quotient when $x^6 - 3$ is divided by $x + 1$...

| **Baseline (TOVA)** | **Ours (AMS)** |
|---|---|
| ...let me compute $f(-1)f(-1)f(-1)$ where $f(x) = x^6 - 1$.<br>**[Problem Drifting & Repetition Collapse]**<br>Wait, no, the original polynomial is $x^6 - 1x^6 - 1x^6 - 1$?<br>Wait, no, the original polynomial is $x^6 - 1x^6 - 1x^6 - 1$?<br>Wait, no, the original polynomial is $x^6 - 1x^6 - 1x^6 - 1$? ...<br>*(Degenerates into an infinite loop and fails to converge)* | ...The polynomial $x^6 - 3x^6 - 3x^6 - 3$ can be written as:<br>$x^6 + 0x^5 + 0x^4 + 0x^3 + 0x^2 + 0x - 3$.<br>**[Stable State Tracking]**<br>So the coefficients are $1, 0, 0, 0, 0, 0, -3$.<br>*(Maintains the correct anchor constants and proceeds to the final correct quotient)* |

### G.2 Case 2: Premature Overturning

Another deceptive consequence of context fragmentation is *Premature Overturning*. The model may successfully deduce the correct intermediate answer, but because the supporting derivation logic was evicted from the cache, it hallucinates a contradiction. Table 20 demonstrates this behavior on an algebraic simplification task. TOVA reaches the correct intermediate state ($14$) but suddenly overturns its own conclusion, outputting $0$. AMS securely holds the logical chain and confidently boxes the correct final answer.

**Table 20: Case Study 2: Premature Overturning (Algebraic Simplification).** The baseline reaches the correct state but overturns it due to fragmented logic, whereas AMS cleanly converges.

**Question:** Simplify $(u + 4)(u - 1) - (u - 3)(u + 6)$.

| **Baseline (TOVA)** | **Ours (AMS)** |
|---|---|
| ...the constants calculate to $14$.<br>**[Premature Overturning]**<br>Wait, so the correct answer is zero. But in the problem statement, the user wrote: "Simplify $(u + 4)(u - 1) - (u + 4)(u - 1)$."<br>So, the expression is $(u+4)(u-1)-(u+4)(u-1)$. So, that's zero.<br>*(Outputs* $\boxed{0}$*)* | ...Now, let's combine like terms.<br>First, the $u^2$ terms: $u^2 - u^2$. That cancels out to $0$.<br>Next, the $u$ terms: $3u - 3u$. That also cancels out to $0$.<br>**[Clean Convergence]**<br>Finally, the constant terms: $-4 + 18 = 14$.<br>*(Outputs* $\boxed{14}$*)* |

# H Complexity and System Efficiency

Compared to standard training-free KV compressors, AMS adds only lightweight bookkeeping on top of the base scorer. For a cache of length $T$ and $H_{kv}$ heads, computing the quality mass requires a single pass over recent-window scores and a prefix sum per head, both $O(H_{kv}T)$ operations. Segment construction via searchsorted and the split and merge heuristics is also linear in $T$. Quota allocation consists of a few vector operations over the number of segments, which is typically one to two orders of magnitude smaller than $T$.

In practice, the dominant cost in decoding-time compression remains the base scoring function (e.g., expected-attention statistics) and the attention computation itself. As shown in our memory evaluation (Appendix H.1), AMS matches the peak memory footprint of existing KV-cache compression baselines when run with the same target cache size. Empirically, Appendix H.2 shows that AMS matches or slightly improves the wall-clock decoding latency of existing gather-based policies, supporting its practical utility in the evaluated single-request setting.

## H.1 Memory Evaluation

We measure peak GPU memory usage during decoding on MATH500 with DEEPSEEK-R1-DISTILL-QWEN-7B, using the same evaluation protocol as in Section 4.1. For each method, we fix the per-layer target cache length $T_{\text{keep}}$ and compression interval, and record the maximum allocated memory (in GB) over the full run. We focus on representative, strong training-free baselines covering streaming-style eviction (StreamingLLM), token-wise attention-based scoring (TOVA, KeyDiff), fixed-structure allocation (ChunkKV-Expected), layer- or head-adaptive allocation (PyramidKV, AdaKV-ExpE2), and reasoning-oriented compression or scoring (R-KV, RPC).

For all methods that explicitly compress and gather the KV cache to the same effective length $T_{\text{keep}}$ (StreamingLLM, TOVA, PyramidKV, and AMS-Expected), the peak GPU memory lies in a narrow band around 15 GB at both $T_{\text{keep}} = 512$ and $T_{\text{keep}} = 1024$. This confirms that, once the target physical cache size is fixed, the memory footprint is essentially determined by $T_{\text{keep}}$, and the additional bookkeeping introduced by AMS is negligible in practice.

In contrast, AdaKV-ExpE2 follows a masking-based allocation strategy that does not physically remove KV entries. Although it enforces an effective budget through attention masks, it keeps (almost) the full KV cache in memory and reaches about 39 to 40 GB peak memory in both settings, more than $2.5\times$ higher than the gather-based KV-compression methods. Compared to AdaKV-ExpE2, AMS-Expected saves roughly 25 GB of peak memory while maintaining, and often improving upon, its pass@1 accuracy, making it a much more memory-efficient choice under the same backbone and decoding setup.

## H.2 Latency Evaluation

Table 8 shows the average wall-clock time per sample on MATH500 with DEEPSEEK-R1-DISTILL-QWEN-7B under decoding-time KV compression at target cache sizes $T_{\text{keep}} \in \{128, 512\}$. We reuse the setup from Section 4.1: for each method, we fix the compression interval and KV budget, run a full pass over the test set, and measure the end-to-end decoding latency per example on the same hardware.

At $T_{\text{keep}} = 512$, the baseline methods **StreamingLLM**, **TOVA**, and **PyramidKV** achieve approximately $48.5$, $52.3$, and $51.6$ seconds per sample, respectively, while our **AMS-Expected** runs at about $44.0$ seconds per sample. The masking-based **AdaKV-ExpE2** is the slowest, at roughly $62.3$ seconds per sample. At the tighter budget $T_{\text{keep}} = 128$, the pattern is similar: AMS-Expected remains the fastest method at around $41.4$ seconds per sample, compared to $50.1$ for StreamingLLM, $67.1$ for TOVA, $67.5$ for PyramidKV, and $91.8$ for AdaKV-ExpE2.

Overall, AMS-Expected matches or improves the latency of existing gather-based KV-compression policies, while AdaKV-ExpE2 incurs a substantially higher runtime cost due to its masking-based allocation and expected-attention scoring. Together with the memory measurements in Appendix H.1, these results show that AMS achieves a strictly better accuracy and memory trade-off.

Crucially, standard fixed-step latency metrics do not capture the full efficiency picture of long-context reasoning. As analyzed in Section C, token-wise methods like TOVA are highly susceptible to

*Repetition Collapse* when the context fragments. To quantify the actual end-to-end efficiency, we measured the generation dynamics under free generation (Table 9).

Because AMS effectively suppresses logical deadlocks and infinite loops, it significantly reduces the overall repetition rate (from 21.58% to 16.18%). Consequently, **AMS-TOVA** produces fewer useless hallucinated tokens and achieves a **faster end-to-end free generation time** (63.85s) compared to the underlying **TOVA** baseline (67.34s).

**Per-token latency and recompression overhead.** To separate computational overhead from differences in generated sequence length, we additionally measure token-normalized latency on the same 50 MATH500 samples. TOVA and AMS-TOVA achieve mean per-sample token-normalized latencies of 25.28 and 25.23 ms/token, respectively. In a separate fixed-600-token, single-sample profiling experiment, one recompression event takes 110.5 ms for TOVA and 330.9 ms for AMS-TOVA. The incremental cost of AMS is approximately 220.4 ms per recompression event, corresponding to 0.43 ms/token when amortized over a 512-token recompression interval. These measurements show that AMS introduces additional periodic recompression work while maintaining comparable observed per-token generation latency in the evaluated single-request setting.

## NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The main claims in the abstract and introduction are supported by the method description in Section 3 and the empirical results in Section 4. The paper frames AMS as a decoding-time KV compression allocation layer, and the reported claims are limited to the evaluated benchmarks, budgets, backbones, and base scorers.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: The paper includes a dedicated Limitations paragraph in Appendix A, discussing the training-free nature of AMS, its dependence on attention-derived mass signals, and the need for broader evaluation on additional architectures, multi-turn settings, and production serving systems.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.

- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

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

   Answer: [N/A]

   Justification: The paper does not present formal theoretical results, theorems, or proofs. The mathematical expressions are used to define the AMS allocation mechanism rather than to establish theoretical guarantees.

   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: The paper provides the main experimental settings in Section 4.1, including benchmarks, models, baselines, cache budgets, and compression intervals. Additional dataset protocols, baseline registry details, AMS implementation details, and hyperparameters are provided in Appendix B.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.

- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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

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

   Answer: [Yes]

   Justification: We provide code and configuration files in the https://github.com/EIT-NLP/AdaptiveMassSegment, together with instructions for reproducing the main AMS and baseline experiments. The evaluated datasets and backbone models are publicly available, and the appendix describes the data access and evaluation protocol.

   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: The experimental setting is described in Section 4.1, including benchmark selection, backbone models, baseline methods, cache budgets, and compression schedule. Further implementation details, baseline configurations, and hyperparameter choices are provided in Appendix B.

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: [No]

Justification: The paper reports deterministic pass@1 results under fixed decoding settings and evaluates full benchmark splits where applicable, but it does not currently include error bars, confidence intervals, or statistical significance tests. Repeated long-form decoding runs across all baselines and budgets would substantially increase the computational cost.

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: The paper reports the hardware and runtime environment in Appendix F, including the use of NVIDIA A800 GPUs, GPU memory, the number of GPUs used per evaluation job, and the main software versions.

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: The research conforms to the NeurIPS Code of Ethics. It develops and evaluates a decoding-time KV compression method on public benchmarks and publicly available backbone models, without involving human subjects, private data, or sensitive personal information.

   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 includes an Impact Statement discussing the positive impact of reducing KV-memory cost for long-context inference. It also acknowledges that more efficient long-form generation may lower deployment costs and should therefore be used with the same safety, monitoring, and evaluation practices as the underlying language model.

    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 generative model, dataset, or deployed system that would require additional safeguards. AMS is a decoding-time KV compression method evaluated on existing public models and benchmarks.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: Appendix B.3 summarizes the publicly available datasets, backbone models, software frameworks, and baseline assets used in this paper, together with their usage and license or terms-of-use status where publicly available. All existing assets are used for research evaluation and are cited in the paper.

    Guidelines:

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

13. **New assets**

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

    Answer: [Yes]

    Justification: The new asset introduced by the paper is the AMS implementation and experimental configuration files, which are provided in the https://github.com/EIT-NLP/AdaptiveMassSegment. The release is intended for research reproducibility and does not include new datasets or pretrained model weights.

Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing, user studies, annotation by human participants, or research with human subjects. All experiments are conducted on existing public benchmarks and machine-generated model outputs.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects, so IRB approval or an equivalent institutional review process is not applicable.

    Guidelines:

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

16. **Declaration of LLM usage**

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

    Answer: [N/A]

Justification: The core method development does not use LLMs as an important, original, or non-standard component of the research process. LLMs are used as evaluated backbone models in the experimental setting, which is described in Section 4.1.

Guidelines:

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