SkipKV: Selective Skipping of KV Generation and Storage for Efficient Inference with Large Reasoning Models

Jiayi Tian[*1,2] **Seyedarmin Azizi**[3] **Yequan Zhao**[1] **Erfan Baghaei Potraghloo**[3] **Sean McPherson**[2] **Sharath Nittur Sridhar**[2] **Zhengyang Wang**[1] **Zheng Zhang**[1] **Massoud Pedram**[3] **Souvik Kundu**[2]

Abstract

Large reasoning models (LRMs) often incur significant key-value (KV) cache overhead, due to their linear growth with the verbose chain-of-thought (CoT) reasoning. This incurs both memory overhead and throughput bottlenecks, limiting efficient deployment. To reduce KV cache size during inference, we first investigate the effectiveness of existing KV cache eviction methods for CoT reasoning. Interestingly, we find that due to unstable token-wise scoring and reduced effective KV budget caused by padding, state-of-the-art (SoTA) eviction methods fail to maintain accuracy in multi-batch settings. Additionally, these methods often generate longer sequences than the original model without eviction, as semantic-unaware token-wise eviction leads to repeated revalidation during reasoning. To address these issues, we present **SkipKV**, a ***training-free*** KV compression method that performs selective *eviction* and *generation*, operating at a coarse-grained, sentence-level sequence removal for efficient CoT reasoning. In specific, it introduces a *sentence-scoring metric* to identify and remove highly similar sentences while maintaining semantic coherence. To suppress redundant generation, SkipKV dynamically adjusts a steering vector to update the hidden activation states during inference, enforcing the LRM to generate concise responses. Extensive evaluations on multiple reasoning benchmarks demonstrate that SkipKV achieves up to $\mathbf{26.7}\%$ higher accuracy compared to baseline methods, at a similar compression budget. Additionally, compared to SoTA, SkipKV yields up to $\mathbf{1.6}\times$ shorter generation length while improving throughput by up to $\mathbf{1.7}\times$. Our code is released at: https://github.com/TTTTTTris/SkipKV.

1 Introduction

Large language models (LLMs) have demonstrated remarkable capabilities in complex reasoning with advancements enabling them to perform multi-step mathematical derivations (Trinh et al., 2024; Luo et al.; Ahn et al., 2024) and code generation (Wang & Chen, 2023; Zhong & Wang, 2024; Mahbub et al., 2026). However, reasoning-oriented models (e.g., DeepSeek-R1 (Guo et al., 2025)) face a critical deployment bottleneck: their tendency to generate lengthy and often redundant reasoning traces that lead to unsustainable memory demands (Chen et al., 2025b). In particular, the large token count linearly increases the key-value (KV) cache memory of the autoregressive large reasoning models (LRMs), often making them the dominant component for large reasoning depth. For instance, a DeepSeek-R1-Distill-Llama-8B model may generate over 32K tokens when solving a single complex math problem; for a batch size of 10, it needs $\sim 2.5\times$ larger KV cache memory compared to the model weights. In addition to the significant memory overhead, the growing KV cache impacts the throughput of the memory-bound decoding stage of LRMs. This highlights the need for reasoning KV cache compression an important research paradigm for long chain-of-thought (CoT) reasoning.

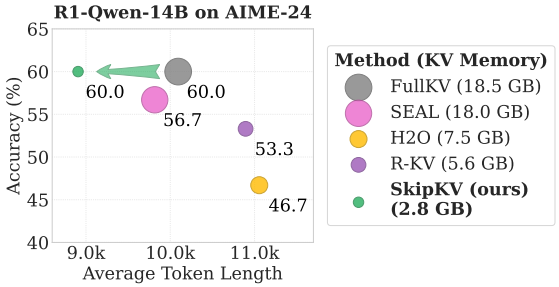

Figure 1. Comparison of KV cache eviction methods. Marker size denotes KV memory usage. SkipKV yields shorter generation length while maintaining high accuracy under a smaller KV budget.

Despite significant research, most of the existing KV compression methods (Zhang et al., 2023; Xiao et al.; Li et al., 2024; Tang et al.) target the KV compression of long-context prefill and lose their efficacy in long CoT tasks for LRMs. For example, H2O (Zhang et al., 2023) improves

[1]University of California, Santa Barbara [2]Intel Labs [3]University of Southern California. Correspondence to: Souvik Kundu <souvikk.kundu@intel.com>.

Proceedings of the 9^{th} MLSys Conference, Bellevue, WA, USA, 2026.

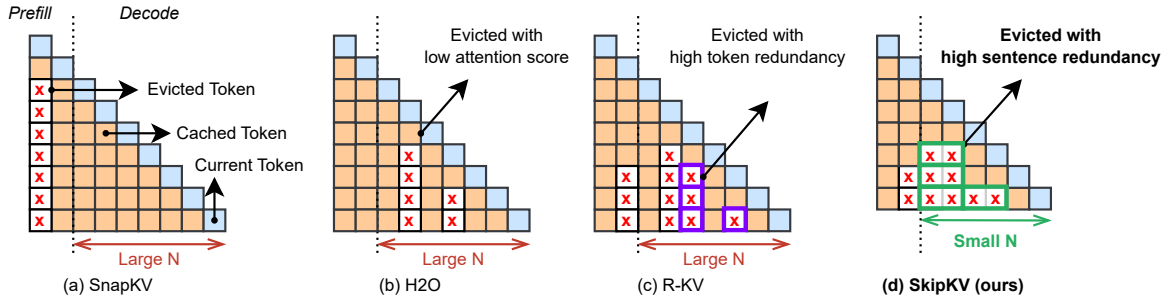

Figure 2. Comparison of KV cache eviction methods during token generation. Cached tokens marked with × indicate evicted positions. (a) SnapKV performs one-time eviction after prefill; (b) H2O evicts tokens with low cumulative attention scores; (c) R-KV prunes redundant tokens based on token-level similarity (purple); (d) SkipKV (ours) groups tokens within sentences (green) to evict high sentence-redundancy regions, achieving high accuracy and shorter generation length (N).

decoding efficiency by retaining a compact subset of KV pairs based on cumulative historical attention scores. Yet, its relies solely on attention magnitude and overlooks semantic coherence across multi-step reasoning spans. SnapKV (Li et al., 2024) leverages attention-based importance estimation to compress the KV cache during the prefill phase, achieving strong performance in long-context scenarios but shows a significant accuracy drop for CoT reasoning tasks. Only recently R-KV (Cai et al., 2025) highlighted a potential limitation of these works, by focusing on *repetitive and redundant* CoT tokens. It then proposed a redundancy-aware metric to effectively remove repetitive tokens along the reasoning path, achieving over 80% KV-cache compression while maintaining strong accuracy. However, despite good accuracy on single-batch settings, R-KV often suffers from a significant accuracy drop for multi-batch settings, limiting its scalable deployment. Moreover, its token-level eviction granularity disregards higher-level semantic structure, often resulting in unnecessarily prolonged reasoning paths. Fig. 2 compares representative KV-cache eviction strategies. Here, we focus on permanent eviction methods, where evicted tokens are no longer accessible in subsequent decoding steps, thereby yielding genuine memory savings.

Our Contributions. To address these limitations, we first investigate the limitations of existing token eviction methods. Our analysis reveals that incoherent token eviction can result in unstable reasoning with reduced accuracy and prolonged generation. Specifically, we find that the fragmented token histories often induce overthinking and force the CoT to generate more tokens for a fixed KV budget. Based on these insights, we then propose **SkipKV**, a sentence-aware selective KV eviction and generation framework designed for efficient CoT reasoning. SkipKV not only preserves reasoning coherence but also *achieves a better trade-off between accuracy and generation length at a fixed KV cache memory budget* (Fig. 1). At its core, SkipKV consists of two key components, namely the **sentence-primary scoring method** and the **adaptive steering method** to efficiently skip token storage and generation, respectively. Specifically, it relies on a *sentence-primary scoring metric* that enables selective KV storage skipping while preserving the semantic coherence of the reasoning process. The **adaptive steering mechanism** suppresses uninformative or redundant sentences during the generation. Additionally, SkipKV adopts a **batch grouping policy** that reduces the number of padding tokens, thereby freeing KV-cache space for valid tokens and improving stability and consistency in multi-batch reasoning with KV eviction.

To demonstrate the effectiveness of SkipKV, we conduct comprehensive evaluations on DeepSeek-R1-Qwen-7B, R1-Qwen-14B, and R1-Llama-8B LRMs across four reasoning benchmarks: AIME-24, LiveCodeBench, MATH-500, and GSM8K. As shown in Fig. 1, SkipKV outperforms state-of-the-art (SoTA) methods, achieving $\mathbf{6.7}\%$ higher accuracy and $\mathbf{22}\%$ shorter generation lengths, with $\mathbf{2}\times$ KV memory compression on R1-Qwen-14B evaluated on AIME-24.

2 Related Works

KV Cache Compression Methods. To mitigate the growing KV cache memory footprint, existing research on *inference-time* KV cache compression can be broadly classified into two categories: KV cache *eviction* (Zhang et al., 2023; Li et al., 2024; Behnam et al., 2025; Tang et al.) and *quantization* (Liu et al., 2024; Kang et al., 2024; Ramachandran et al., 2026).

The KV eviction methods that *permanently remove redundant tokens*, including H2O (Zhang et al., 2023) and SnapKV (Li et al., 2024), primarily rely on the token importance ranked based on attention scores to remove low-scoring KV tokens and meet a fixed KV budget, reducing the total memory demand. Chunk-based eviction methods, such as ChunkKV (Liu et al., 2025c), extend this idea by grouping tokens into chunks and performing eviction at the chunk level, which reduces eviction overhead and improves memory locality while still relying on attention-derived importance signals. These works, while performing well on

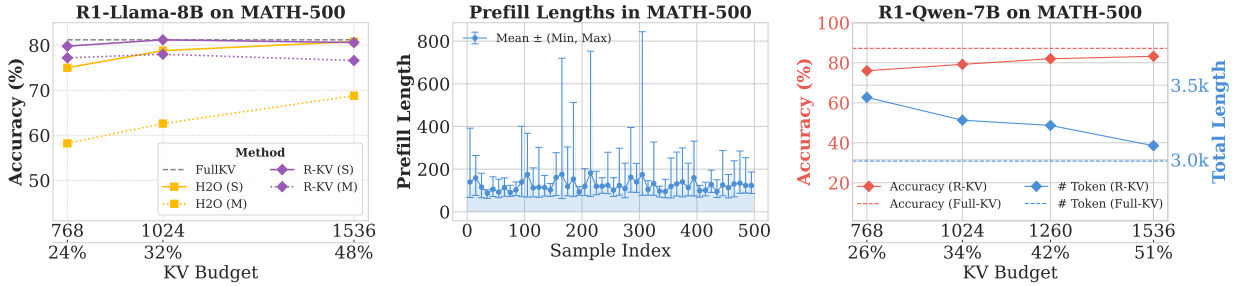

Figure 3. **Left:** Accuracy comparison for single- and multi-batch decoding of H2O (Zhang et al., 2023) and R-KV (Cai et al., 2025). **Center:** Visualization of the prefill token length distribution of MATH-500, and the min-max range of each batch (batch size, bs = 10). **Right:** Accuracy and generated token length versus KV budget with R-KV eviction on MATH-500 (bs = 10).

non-reasoning benchmarks with large prefill length, fail to demonstrate good accuracy for CoT compression. Only recently, a contemporary work, R-KV (Cai et al., 2025) identifies a key limitation of these methods: *not adhering to the redundancy in the token importance scoring*. R-KV proposes a redundancy-aware token scoring to yield SoTA compression-accuracy trade-off for CoT reasoning. However, R-KV still suffers from a larger generation length and reduced accuracy in multi-batch decoding.

Other selection-based eviction methods, such as Quest (Tang et al.), keep FullKV in global memory and bring a fractional chunk into local memory that remains relevant to the query. While such approaches can preserve accuracy by maintaining access to the complete historical context, the KV cache still grows linearly with sequence length, which can limit decoding throughput, reduce batch size scalability, and lead to out-of-memory issues for long chain-of-thought (CoT) reasoning workloads (Liu et al., 2025a; Hu et al., 2025).

In contrast to eviction, KV quantization reduces memory footprint by storing KV tokens at lower precision. However, for large reasoning models (LRMs), quantization may degrade CoT reasoning performance (Liu et al., 2025b). In this work, we focus on improving KV eviction, while KV quantization remains an orthogonal direction.

Other KV Cache Reduction Methods. Apart from compression, earlier works have explored approaches to reduce the KV memory footprint via different methods, including KV *sharing*, *early exiting*, and *steering*. Sharing methods, like KVsharer (Yang et al., 2024), relies on tensor similarity between KV caches of different layers of a model, to allow them to be shared across ≥ 1 layer. However, this sharing costs a significant accuracy drop on CoT reasoning tasks. DEER (Yang et al., 2026) monitors transition tokens during the reasoning process and estimates the confidence of early termination by inducing trial answer tokens, thereby reducing reasoning length through adaptive early exits. KV steering (Chen et al., 2025a; Azizi et al., 2025) on the other hand introduces a latent-space calibration method to identify and manipulate non-important thoughts in CoT to reduce the final reasoning sequence length. While these methods can reduce the generation token count to improve per-sample memory, they cannot meet a fixed KV budget and their batch-level scalability remains an open problem.

3 Motivational Case Studies

In this section, we analyze the key limitations of the representative SoTA CoT token eviction methods.

> **Observation 1:** *With KV eviction, reasoning accuracy drops in multi-batch decoding compared to that with single-batch.*

Description. The benefit of KV cache eviction methods lies in their ability to reduce KV memory usage, thereby enabling larger batch size with improved generation throughput. To analyze the accuracy robustness of the existing SoTA KV eviction methods (Cai et al., 2025; Zhang et al., 2023) on CoT reasoning tasks, we evaluate the accuracy for a batch size of 1 and 10, respectively. Specifically, we measure the accuracy on the MATH-500 with the R1-Llama-8B at different KV budgets (as a fraction of the KV memory, with FullKV as 100%). As depicted in Fig. 3 (left), the multi-batch decoding leads to substantial accuracy degradation, for both R-KV and H2O, particularly at lower KV budgets. This limitation may be attributed to the *effective KV budget reduction* for the multi-batch scenario due to the padding tokens. These tokens are added to align variable-length prefill sequences across different samples in a batch. Upon enabling a fixed memory budget via KV eviction, the padding tokens consume memory space resulting in a reduction in the effective KV budget, and thus potentially reduce accuracy. In addition, these tokens can also distort attention distributions, leading to unstable token-level importance estimation that can further contribute to reasoning consistency degradation. As depicted in Fig. 3 (center), the prefill token length in MATH-500 varies widely across samples, with some variations exceeding 400 tokens in a batch. This variation leads to excessive padding when sequences are batched together, increasing KV cache waste and exacerbating the performance drop in multi-batch decoding with eviction.

<|User|>Answer the following questions. You should think step-by-step and put your final answer within \boxed{}.
Question: Evaluate $(1+2i)6-3i$.<|Assistant|>
[Re-validate for 8 times and generate 1517 tokens in total]
... So, the result should be 6 + 9i.
Wait, let me double-check... 6 + 9i. That seems correct.

Alternatively, ... Let me write it out again:
... Then, subtracting 3i: 6 + 12i - 3i = 6 + (12i - 3i) = 6 + 9i.

Just to be thorough, maybe I can represent this in another way. ...
Alternatively, I can think of it as 6*(1 + 2i) - 3i.
Another way to approach this is ...
So, 6 + 12i - 3i = 6 + (12 - 3)i = 6 + 9i. Yep, that's consistent.
I think I'm confident that the result is 6 + 9i. Let me... 6 + 9i. That seems reasonable.

Alternatively, ... (6, 9). So, that's another way to see it.
Just to recap:
... - 6 + 12i - 3i = 6 + (12i - 3i) = 6 + 9i

Just to make sure, let me compute it again step by step:
... - 6 + 12i - 3i = 6 + (12i - 3i) = 6 + 9i.

Yep, same result. I think that's solid.
I guess another way ...
So, after all that, I'm pretty confident that the answer is 6 + 9i.
Final Answer
The result of evaluating $(1+2i)6 - 3i$ is \boxed{6 + 9i}.

Figure 4. SoTA token-based eviction (Cai et al., 2025) often selects fragmented tokens from the final answer $(6 + 9i)$ (highlighted in orange boxes), causing repeated self-validation and non-execution thoughts (highlighted in yellow). Tokens in blue are retained, while those in gray are evicted.

> **Observation 2:** *At reduced KV budgets the total generation length often increases compared to models without any KV compression.*

Description. Fig. 3 (right) shows the accuracy and average token length (continuous lines) with DeepSeek-R1-Qwen 7B for a batch size of 10, evaluated on MATH-500. In addition to reduced accuracy, models with R-KV compression consistently generate longer sequences across all KV budgets compared to models without compression. This observation indicates that token eviction can cause the model to extend its reasoning chain and generate longer outputs. This may be attributed to the loss of contextual information in the KV cache: when portions of the valid context are removed, the model may compensate by regenerating reasoning steps or adding redundant segments to recover missing information. The longer generations result in increased computational cost due to additional forward passes, thereby offsetting efficiency gains from cache size reduction.

> **Observation 3:** *Token-level eviction often causes fragmented removal of words, leading the LRM to overthink.*

Description. To further investigate the cause of increased generation length, we visualize the tokens retained after R-KV eviction in Fig. 4. We select a sample from MATH-500 and show model outputs where the retained tokens are highlighted in blue and the evicted ones in gray.

The limitations here can be summarized in two ways. First, eviction purely based on token-level redundancy scores (Cai et al., 2025) often removes numbers from crucial mathematical computation steps, thereby disrupting the reasoning flow. Second, token-level eviction may often retain fragmented tokens from the correct answer, misleading the LRM to re-validate partial results and generate unnecessarily long or uncertain reasoning chains. For instance, in Fig. 4, the tokens retained by R-KV frequently include disjoint numerical entities from intermediate reasoning steps (e.g., $6 + 12i - 3i \rightarrow +2i$) or from the final answer (e.g., $(6, 9) \rightarrow (, 9)$). Such fragmented retention causes the model to repeatedly re-derive or re-check parts of the answer, resulting in longer yet redundant reasoning trajectories.

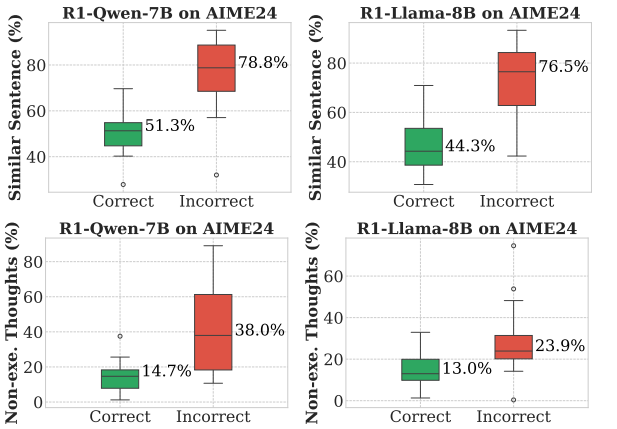

Figure 5. Statistics on the ratio of high-similarity sentences (top) and non-execution thoughts (bottom) for samples correctly and incorrectly answered by the models.

These findings highlight the need for a coherent eviction policy that operates on higher-level semantic units (e.g., sentences or reasoning segments) to achieve a better trade-off between accuracy and generation length.

4 SkipKV: Methodology

In this section, we first analyze the sentence-level properties of reasoning traces motivating the design of SkipKV (illustrated in Fig. 6). We then present the two core components of SkipKV: (1) sentence-level skipping of the KV cache storage guided by semantic redundancy scoring; and (2) adaptive KV steering for controlled generation to suppress unnecessary thought expansion. Finally, we present a simple yet effective batch-grouping-driven eviction strategy to enhance the accuracy robustness in multi-batch decoding.

Definition: Pairwise Sentence Similarity (PSS) *between two sentences is defined as the measured cosine similarity between the vector embeddings of the two. Let the vector embedding of two sentences be* $\mathbf{v}_i, \mathbf{v}_j \in \mathbb{R}^d$*, respectively. The PSS score is computed as*

$$\mathrm{PSS}(\mathbf{v}_i, \mathbf{v}_j) = \mathbf{v}_i^\top \mathbf{v}_j. \quad (1)$$

To understand the differences between successful and failed reasoning trajectories from the semantic perspective, we perform a fine-grained analysis of sentence- (thought-) level properties in the model generated output with FullKV. We

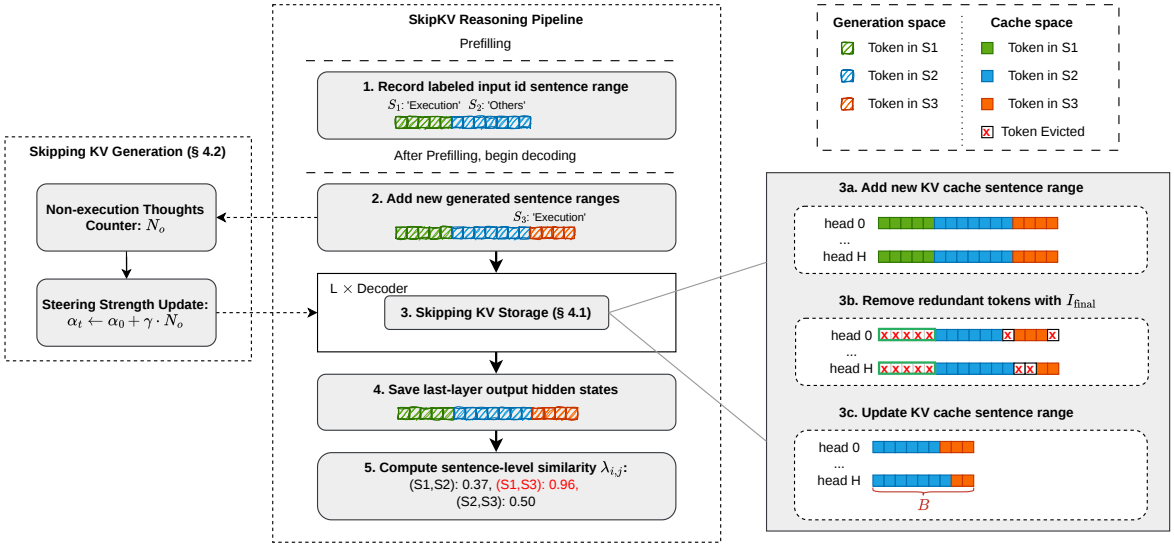

Figure 6. Overview of the SkipKV framework. It selectively skips KV-cache storage and generation by sentence-level redundancy detection. The central reasoning pipeline illustrates the end-to-end process from prefill to decoding, where input sentence ranges and types are recorded, sentences are scored, and the KV cache is compressed. The left panel depicts the adaptive steering mechanism used to skip KV generation (§ 4.2), while the right panel illustrates sentence-level KV storage skipping (§ 4.1), including the KV range monitoring logic (3a, 3c) and the sentence-oriented eviction strategy (3b).

focus on two aspects: (i) the semantic similarity among generated sentences, and (ii) the ratio of non-execution (less important) thoughts. Notably, following the definition of (Chen et al., 2025a), here we classify the generated tokens as **execution** (important tokens relevant to the actual answer) and **non-execution** (less important tokens generated to verify the response via reflection or transition) thoughts.

> **Observation 4:** *Both correct and incorrect reasoning responses generate highly similar sentences with the latter scenario usually generating higher % of similar sentences.*

Description. We perform an experiment with R1-Qwen-7B and R1-Llama-8B on AIME24 to compute the PSS. In particular, we first collect model output texts and segment them into individual sentences separated by newline and punctuation-based patterns (e.g., '\n\n'). Each sentence is then encoded into a semantically meaningful embedding vector using a BERT-based sentence transformer (Reimers & Gurevych, 2019). We then compute the PSS for all the sentence-embedding pairs and mark pairs with a PSS score ≥ 0.95 as highly similar, yielding a measure of semantic redundancy. Interestingly, as shown in Fig. 5 (top), incorrect responses consistently exhibit up to $1.7\times$ higher % of similar sentences as compared to correct responses.

> **Observation 5:** *Incorrect responses generate significantly higher % of non-execution thoughts compared to the correct ones.*

Description. As shown in Fig. 5 (bottom), incorrect generations consistently exhibit higher ratios of non-execution thoughts compared to correct responses. In particular, when evaluating on AIME24 with R1-Qwen-7B and R1-Llama-8B, respectively, we observe around $2.6\times$ and $1.8\times$ higher % of non-execution thoughts in incorrect outputs compared to correct outputs. These patterns indicate that when models fail, they tend to produce repetitive or stagnant reasoning steps, revisiting semantically similar content or generating meta-level commentary rather than performing concrete problem-solving actions.

4.1 Skipping KV Storage with Sentence-level Scoring

Here, we first introduce the sentence-level similarity scoring and then present the cumulative scoring mechanism to decide which sentences and tokens to evict.

Sentence Similarity Score. We use this score to compute the sentence-level redundancy. However, using a sentence transformer to generate the latent vector representation of the sentences is impractical as it costs significant compute overhead during decoding. Instead, we leverage the last-layer hidden state, denoted as $\mathbf{H} \in \mathbb{R}^{bs \times N \times d}$, as latent contextual representations of the sentence segments. Specifically, for each sample in a batch, we identify the start and end indices of each sentence segment by partitioning the total sequence (N) into small chunks that are separated by punctuation-based delimiters (Step 1–2 in Fig. 6). For a sentence i at batch ID k, we then compute its vector embedding

$\mathbf{v}_i \in \mathbb{R}^{1\times1\times d}$ as the mean of the vectors in that sentence,

$$\mathbf{v}_i = \text{mean}\,(\mathbf{H}[k]_{b_i:e_i}). \tag{2}$$

Here, b_i and e_i are the beginning and end indices of sentence i. We then compute the PSS following Eq. (1), and define the redundant sentence set as

$$\begin{aligned}\mathcal{P} &= \{i : \lambda_{i,j} > \tau, i \le j\},\\ &\text{where } \lambda_{i,j} = \mathbf{v}_i^\top \mathbf{v}_j.\end{aligned} \tag{3}$$

This means that if a pair (i, j) exceeds a predefined threshold τ (e.g., 0.95), the earlier sentence i is flagged as redundant, while the later one j is retained.

Token Importance Score. Let $\mathbf{Q} \in \mathbb{R}^{bs\times H_q\times\alpha\times d}$ denote the observation window of recent α query tokens (Li et al., 2024), and $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{bs\times H_k\times N'\times d}$ denote the current key and value cache, respectively. Here, H_q and H_k represent the number of query and key-value heads.Most LRMs use group-query attention, where for each query head $i \in \{1, \ldots, H_q\}$, its associated key-value head is denoted by $g(i) \in \{1, \ldots, H_k\}$, where $g(i) = \lfloor \frac{i}{n} \rfloor$. Here, n is an integer that defines the number of query heads over which each K/V is shared. The attention importance for each query head i in a batch is denoted as $\mathbf{A}^i \in \mathbb{R}^{bs\times1\times\alpha\times N'}$, where

$$\mathbf{A}^i = \text{softmax}(\frac{\mathbf{Q}^i\mathbf{K}^{g(i)\top}}{\sqrt{d}} + \mathbf{M}) \tag{4}$$

Here, $\mathbf{M}$ denotes the attention mask used for multi-batch decoding. We then compute the attention importance for a key head h_k as $\mathbf{I}^{h_k}$ = softmax(maxpool($\mathbf{A}^{h_k\cdot n}$: $\mathbf{A}^{h_k\cdot n+(n-1)}$)). We normalize the attention importance matrix on the observation window of α, for each head, to produce the token importance matrix $\mathbf{I}_\alpha^{h_k} \in \mathbb{R}^{bs\times N'}$.

Token Redundancy Score. Inspired by R-KV, we summarize the token redundancy removal method. For each key-value head h_k, we define the key state as $\mathbf{K}^{h_k} \in \mathbb{R}^{bs\times1\times N'\times d}$, and the token redundancy $\mathbf{R}^{h_k} \in \mathbb{R}^{bs\times N'}$ is calculated as

$$\begin{aligned}\mathbf{R}^{h_k} &= \text{mean}\left(\text{softmax}(\bar{\mathbf{K}}^{h_k}\bar{\mathbf{K}}^{h_k\top})\right),\\ &\text{where } \bar{\mathbf{K}}^{h_k} = \frac{\mathbf{K}^{h_k}\odot\mathbf{M}}{||\mathbf{K}^{h_k}\odot\mathbf{M}||_2+\epsilon}.\end{aligned} \tag{5}$$

Here, we also consider the attention mask $\mathbf{M}$ to reduce the influence of the padding tokens on the redundancy score.

Sentence Redundancy Driven Cumulative Score. We then formulate the overall eviction score for KV compression (Steps 3b in Fig. 6) by combining the token importance score $I_\alpha^{h_k}$, token redundancy score R^{h_k}, and sentence similarity score $\lambda_{i,j}$:

$$I_{\text{final}} = \begin{cases}\sigma I_\alpha^{h_k} - (1-\sigma)R^{h_k} - \lambda_{i,j}, & \text{if } i \in \mathcal{P},\\ \sigma I_\alpha^{h_k} - (1-\sigma)R^{h_k}, & \text{otherwise.}\end{cases} \tag{6}$$

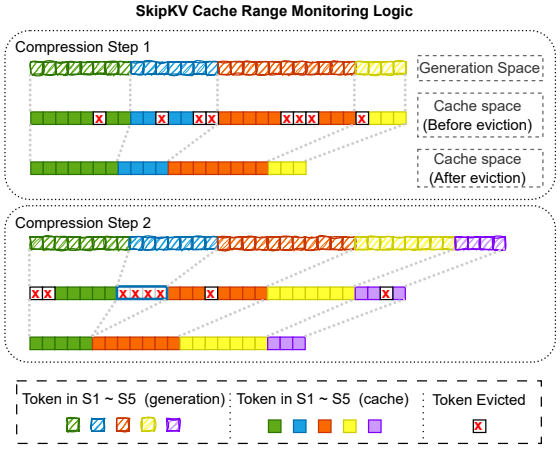

Figure 7. Illustration of the cache range monitoring process over two consecutive compression steps. Colored blocks represent distinct sentence spans in the generation space and their corresponding regions in the KV cache. Gray dashed lines indicate the mapping of sentence range.

σ controls the trade-off between token importance and redundancy. As described earlier, $\lambda_{i,j}$ is the sentence redundancy score associated with sentence pair (i, j), shared over all tokens of the i^{th} sentence. To meet a specific token budget, we evict tokens in increasing order of final score I_{final}. Importantly, *since similarity scores for redundant sentences ($\geq$ 0.95) are typically an order of magnitude higher than token-level scores ($\sim$ 0.1), Eq. (6) ensures that highly redundant sentences are removed before token level eviction.*

KV Cache Sentence Range Monitoring Logic. To ensure that sentences in the redundancy set $\mathcal{P}$ are evicted consistently from the cache space, we introduce a *KV-cache range monitoring mechanism* (Steps 3a and 3c in Fig. 6). This mechanism is defined by a mapping function Φ, that converts the i^{th} sentence span in the original generation space (gs) to that in the cache space (cs), $\Phi(b_i^{(gs)}, e_i^{(gs)}) \rightarrow (b_i^{(cs)}, e_i^{(cs)})$. To correctly associate sentence-level scores for each sentence in $\mathcal{P}$, we record the token span ids $(b_i^{(gs)}, e_i^{(gs)})$ and the corresponding PSS in a lookup table $\mathcal{T}$ as

$$\mathcal{T} \leftarrow \{(b_i^{(gs)}, e_i^{(gs)}) \mapsto \lambda_{i,j}\}, \quad \forall i \in \mathcal{P}. \tag{7}$$

By applying the mapping function Φ, we obtain the **cache-aligned lookup table** as

$$\Phi(\mathcal{T}) \rightarrow \{(b_i^{(cs)}, e_i^{(cs)}) \mapsto \lambda_{i,j}\}, \quad \forall i \in \mathcal{P}. \tag{8}$$

The cache-aligned lookup table $\Phi(\mathcal{T})$ is then used in Eq. (6) to determine whether $i \in \mathcal{P}$ during eviction.

As illustrated in Fig. 7, the sentence range monitoring logic maintains a dynamic mapping between the generation space (gs) and the cache space (cs) throughout the compression process. After compression step 1, the retained tokens serve

as the initial cs tokens for step 2, with the mapping continuously updated to correctly align gs indices with their corresponding positions in cs.

Updating Cache Ranges Before Eviction. As eviction is performed periodically during decoding, newly generated tokens—often initiating or completing sentences—may be appended to the gs tokens at each eviction step. Accordingly, their corresponding sentence spans are also added to the cs, following two cases: (1) initialization of sentence ranges during the first step (Fig. 7, top), and (2) appending new ranges in subsequent steps (Fig. 7, bottom). In case (1), the cache ranges are simply initialized to those in the gs:

$$(b_i^{(cs)}, e_i^{(cs)}) \leftarrow (b_i^{(gs)}, e_i^{(gs)}), \forall i \tag{9}$$

At compression step t, we denote the *generation length* and *cache length before eviction* as $l_t^{(\text{gs})}$ and $l_t^{(\text{cs})}$, while the *post-eviction cache length* is constrained by the budget B. In case (2), for each new sentence added during the current compression step, we sequentially update the sentence range in increasing order of sentence index. For the i^{th} one, it is,

$$\begin{aligned} b_i^{(\text{cs})} &\leftarrow e_{i-1}^{(\text{cs})} + 1, \\ e_i^{(\text{cs})} &\leftarrow l_t^{(\text{cs})} - \Delta, \text{where } \Delta = l_t^{(\text{gs})} - e_i^{(\text{gs})}. \end{aligned} \tag{10}$$

The end position $e_i^{(\text{cs})}$ is determined by subtracting the offset Δ from the current cache length $l_t^{(\text{cs})}$. Here, Δ represents the *length of residual tokens* in the gs starting from the $(i+1)^{\text{th}}$ sentence. This offset is equivalent to the length of the newly appended tokens and can be computed as the difference between the total generation length and the end position of the i^{th} sentence in the generation space.

Updating Cache Ranges After Eviction. As shown in Fig. 7, sentence spans must be remapped into the new cache coordinate space according to the surviving token indices in each compression step. Let the set of surviving token indices be $P = \{p_1, p_2, \ldots, p_B\}$ with $0 \le p_1 < p_2 < \cdots < p_B < l_t^{(cs)}$, with p_k denoting the k-th surviving token index. The corresponding cache ranges are then updated as

$$\begin{aligned} b_i^{(\text{cs})} &\leftarrow \min\{k \mid p_k \ge b_i^{(\text{cs})}\}, \\ e_i^{(\text{cs})} &\leftarrow \max\{k \mid p_k \le e_i^{(\text{cs})}\}, \forall i \end{aligned} \tag{11}$$

Here, $b_i^{(\text{cs})}$ is reassigned to the earliest surviving index not earlier than the original start, and $e_i^{(\text{cs})}$ to the latest surviving index not later than the original end. If no tokens in the compressed cache space survive, the corresponding sentence range is discarded from the post-eviction cache space.

4.2 Skipping KV Generation with Adaptive Steering

In addition to removing redundant thoughts after their generation, we further propose to *skip unnecessary thoughts before generation* through an adaptive steering mechanism. For this, we add a steering vector to the hidden state of certain LRM layer(s), enforcing the model to be more precise. However, unlike earlier works on steering (Chen et al., 2025a; Azizi et al., 2025) that use a fixed strength (α_0) for the steering vector throughout generation, we propose a strength adaptation based on the number of non-execution thoughts. The details are presented below.

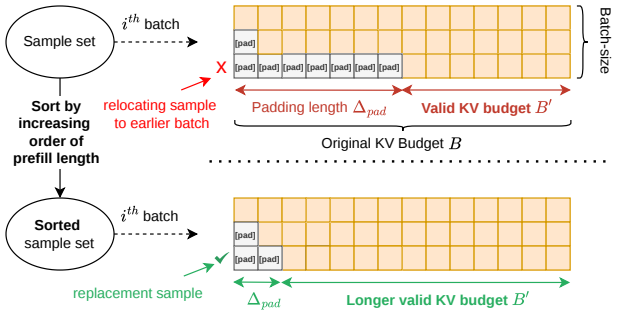

Figure 8. Batch grouping increases valid KV budget for high-performance multi-batch decoding. Yellow blocks: valid tokens in the KV cache, Gray blocks: padding tokens.

Inspired by SEAL (Chen et al., 2025a) we first construct the steering vector using 1000 samples from the MATH training set (Hendrycks et al.), aiming to shift the latent representations towards execution-style reasoning. Such reasoning-oriented steering directions exhibit strong transferability across tasks, as shown in prior work (Chen et al., 2025a; Azizi et al., 2025) and our experiments. Importantly, our objective is not to eliminate non-execution thoughts. While execution thoughts correspond to explicit computation, non-execution thoughts can also be beneficial. We therefore treat this distinction as *statistical rather than causal*, and softly bias the model toward patterns more frequently associated with successful reasoning.

The steering vector is computed as the mean latent representation difference between execution and non-execution thoughts: $\mathbf{v} = \overline{\mathbf{H}_E} - \overline{\mathbf{H}_O}$, where $\overline{\mathbf{H}_E}$ and $\overline{\mathbf{H}_O}$ denote the average hidden states of execution and non-execution thoughts, respectively. During generation, at each reasoning newline delimiter $y_t \in \mathcal{D}$, we inject the steering vector into the hidden representation of the selected layer k, where $\mathcal{D}$ denotes the predefined delimiter set (see Appendix A.1), i.e., $\mathbf{H}_k \leftarrow \mathbf{H}_k + \alpha_t \cdot \mathbf{v}$. Here, α_t denotes the steering strength controlling the degree of latent adjustment toward execution-oriented directions.

We dynamically adjust the steering strength α_t based on the *model's observed reasoning behavior*. Specifically, as illustrated in Fig. 6 (left), we maintain a running counter of non-execution thoughts, denoted as N_o, and update the steering strength as $\alpha_t \leftarrow \alpha_0 + \gamma \cdot N_o$, where α_0 is the initial steering strength and γ is a predefined increment factor con-

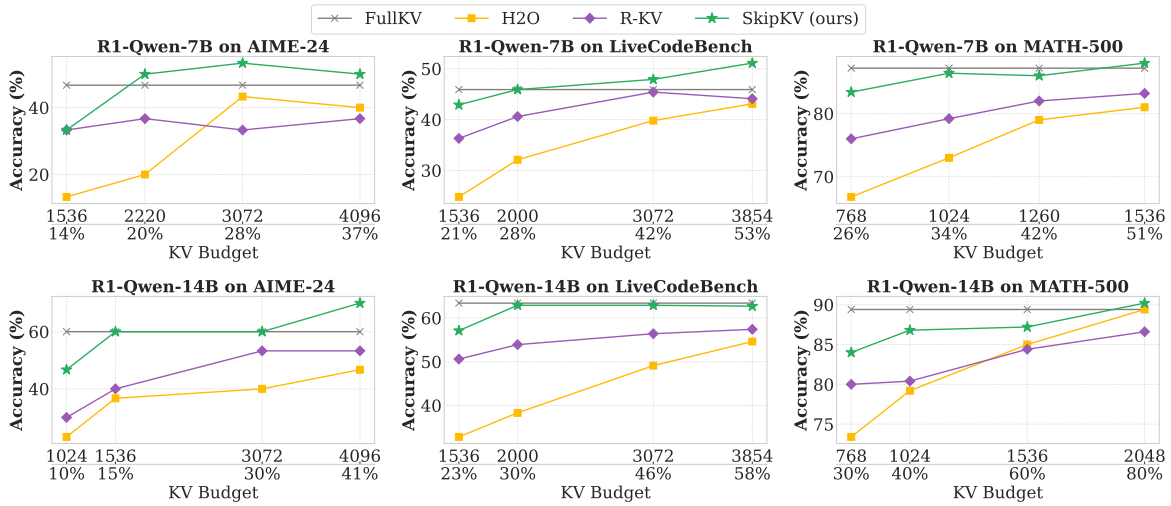

Figure 9. Accuracy comparison under different KV-cache budgets for SkipKV, H2O, R-KV, and FullKV across three reasoning benchmarks and R1-Qwen-7B and 14B models. SkipKV consistently achieves higher accuracy under tighter KV budgets, maintaining full accuracy even at only 15 % KV budget on AIME-24. Results are reported as pass@1.

trolling the degree of steering aggressiveness. In this way, the steering strength for each sample is adaptively adjusted throughout the reasoning process, enabling the model to shorten generation length by selectively suppressing unnecessary thoughts.

4.3 Batch Grouping for Multi-batch Decoding

For each sample in a batch the *valid* KV budget is given by, $B' = B - \Delta_{pad}$, with Δ_{pad} being the sample's padding token count. For any sample, Δ_{pad} is given by,

$$\Delta_{pad} = N_p^{max} - N_p, \tag{12}$$

where N_p and $N_p^{\max}$ denote the prefill length of the current sample and the maximum prefill length within a batch, respectively. When prefill lengths vary significantly across samples, the resulting padding overhead Δ_{pad} can be substantial, reducing the effective KV budget.

To mitigate this inefficiency, we propose a batch grouping strategy that minimizes wasteful padding. Specifically, we first *sort* all samples in increasing order of their prefill lengths, and then *group* them into batches of size bs based on this ordering. As shown in Eq. (12), reducing the gap between $N_p^{\max}$ and N_p lowers the padding overhead, thereby increasing the effective budget B'. Fig. 8 illustrates an example of this sample rearrangement. In §5.3 we empirically validate the effectiveness of increased B' in improving LRM accuracy for multi-batch decoding.

5 Experiments

In this section, we first compare the accuracy and generation length of SkipKV on complex reasoning tasks against recent token-eviction methods across multiple reasoning models and KV-budget ratios. We then compare SkipKV with other efficient reasoning baselines, followed by an evaluation of reasoning throughput and ablation studies on each component of SkipKV, including its impact on the effective KV budget under batch grouping. Additional experimental details and evaluation results are provided in the Appendix.

5.1 Experimental Setup.

Models and Datasets. We conduct evaluations on DeepSeek-R1-Distill-Qwen-7B, DeepSeek-R1-Distill-Qwen-14B, and DeepSeek-R1-Distill-Llama-8B (Guo et al., 2025). Our experiments cover both mathematical reasoning benchmarks—MATH-500 (Lightman et al., 2023), AIME (Mathematical Association of America, 2024), and GSM8K (Cobbe et al., 2021)—and code generation using LiveCodeBench (Jain et al., 2024). We constrain the maximum generation length to 8,192 tokens for GSM8K and MATH-500, 10,000 tokens for LiveCodeBench, and 16,384 tokens for AIME-24. Following SEAL (Chen et al., 2025a), we derive the steering vector using 1,000 samples from the training set of the Math dataset (Hendrycks et al.). Unless otherwise stated, the evaluation batch size is set to 10 for R1-Qwen-7B and R1-Llama-8B, while for R1-Qwen-14B, it is set to the maximum that can fit into GPU memory. All experiments are conducted on a single NVIDIA A100 (40GB) GPU.

5.2 Main Results

Comparison of Accuracy with Eviction Methods. We compare the reasoning accuracy of SkipKV with prior KV cache eviction methods, including H2O, R-KV, and the FullKV baseline across multiple reasoning datasets and model scales, as shown in Fig. 9 and Fig. 13 in Appendix A.3. Fol-

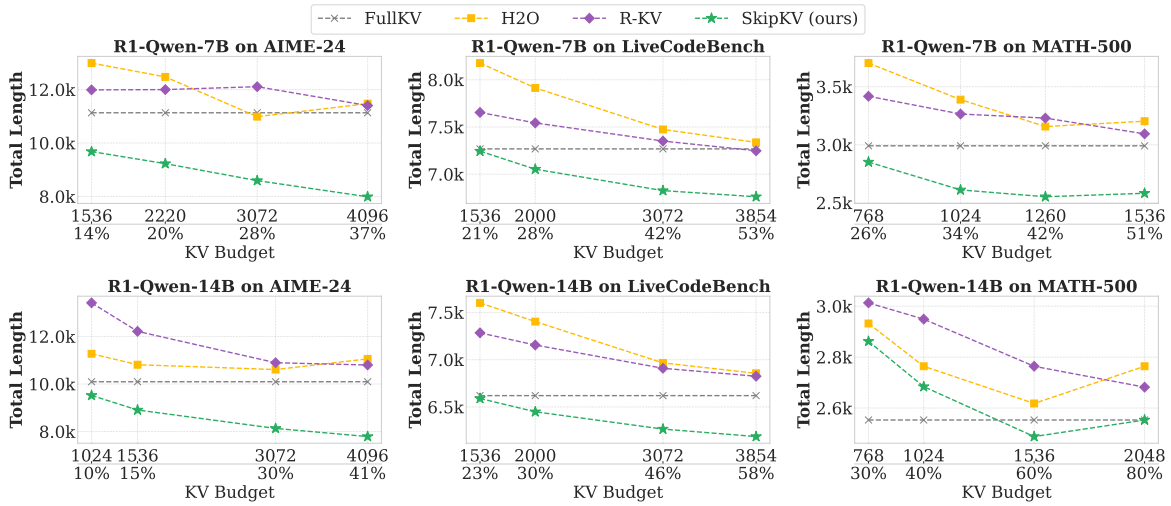

Figure 10. Total generation tokens on SkipKV under different KV budget with H2O, R-KV, and FullKV across three datasets and two models. SkipKV consistently generates fewer tokens and could achieve up to 30% fewer generation length compared with FullKV.

lowing R-KV, we define the KV cache budget ratio as the ratio of the allocated KV cache budget to the average generation token length under FullKV for each model–dataset pair. Different from prior token-level KV eviction methods, which suffer from severe accuracy degradation during multi-batch decoding, our SkipKV approach consistently maintains high accuracy with significantly lower KV memory across all reasoning tasks and models. On challenging reasoning tasks such as AIME-24, SkipKV matches the FullKV accuracy using $6.7\times$ lower KV cache memory on R1-Qwen-14B, while maintaining more than $4\times$ lower KV memory usage across all models. In addition, SkipKV could achieve better performance compared with FullKV on a limited KV budget. For example, SkipKV improves accuracy by 5.2% on LiveCodeBench with $2\times$ less KV memory evaluated on R1-Qwen-7B, and by 10% on AIME-24 while using $2.5\times$ lower KV memory evaluated on R1-Qwen-14B.

Comparison of Token Length with Eviction Methods. Besides achieving higher reasoning accuracy, SkipKV also provides substantial generation efficiency gains by producing fewer tokens. As shown in Fig. 10, prior token-level eviction methods consistently generate more tokens than FullKV across all models and tasks, whereas SkipKV reduces the total token length by up to 28% compared with FullKV. Compared with R-KV, SkipKV generates up to 32%, 39%, and 48% fewer tokens on R1-Qwen-7B, R1-Qwen-14B, and R1-Llama-8B, respectively—translating to a $1.5-2\times$ reduction in generation latency. Additionally, on the MATH-500 dataset, SkipKV produces approximately 15% fewer tokens while using $3\times$ less KV memory than FullKV on R1-Qwen-7B resulting in $3.5\times$ overall memory-latency benefits. Similarly, on LiveCodeBench, SkipKV consistently achieves around 10% shorter generation lengths across all KV budget settings while maintaining comparable accuracy to that with FullKV.

Table 1. Comparison of total latency and throughput under different batch sizes on GSM8K. Evaluated on one A100-40GB.

Metric	Batch-size	10	50	100	120	140
Total Latency (min) ↓	FullKV	324	500	OOM	OOM	OOM
	SEAL	178	192	OOM	OOM	OOM
	R-KV	227	96	66	73	70
	SkipKV (ours)	**136**	**58**	**52**	**66**	**68**
Throughput (samples/min) ↑	FullKV	4.07	2.64	OOM	OOM	OOM
	SEAL	7.41	6.87	OOM	OOM	OOM
	R-KV	5.81	13.7	20.0	18.1	18.8
	SkipKV (ours)	**9.70**	**22.7**	**25.4**	**20.0**	**19.4**

Comparison with Other Efficient Reasoning Methods. We further compare SkipKV with SEAL that focus on reducing generation length. Fig. 11 presents the comparison of KV-cache memory consumption and reasoning accuracy among SkipKV, SEAL, and FullKV on the AIME-24 dataset and across three reasoning models. We set the steering factor of SEAL to 1, following its original configuration. Because SEAL does not explicitly compress the KV cache, its KV memory footprint is approximated using the average number of active tokens per sequence, which is proportional to the mean generation length. While SEAL successfully shortens reasoning traces and maintains accuracy, its modest reduction in output token length (about 10%) translates to only limited KV memory savings. In contrast, SkipKV jointly skips KV generation and storage, achieving up to 13.3% accuracy improvement and $6.6\times$ KV memory reduction when compared with SEAL. These results highlight that SkipKV effectively balances memory compression and reasoning fidelity across different reasoning models.

Throughput Analysis. We evaluate the end-to-end throughput of different methods on the GSM8K dataset using R1-Qwen-7B with a maximum generation length of

Table 2. Ablation of SkipKV evaluated on AIME24 using R1-Qwen-7B. Progressive inclusion of Sentence Scoring, Adaptive Steering, and Batch Grouping improves accuracy and reduces total token length compared with the R-KV baseline.

	Accuracy (%) ↑			Total Token Length ↓		
KV Budget	2220 (20%)	3072 (27%)	4096 (37%)	2220 (20%)	3072 (27%)	4096 (37%)
FullKV	46.7			11132		
R-KV	36.7	33.3	36.7	12000	12109	11403
+ Sentence Scoring	40.0 (+3.3)	36.7 (+3.4)	40.0 (+3.3)	11332 (–6%)	11342 (–6%)	11819 (+4%)
+ Adaptive Steering	40.0 (+3.3)	43.3 (+10.0)	40.0 (+3.3)	8860 (–26%)	10101 (–17%)	10041 (–12%)
+ Batch Grouping (SkipKV)	**50.0 (+13.3)**	**53.3 (+20.0)**	**50.0 (+13.3)**	**9228 (–23%)**	**8593 (–29%)**	**7988 (–30%)**

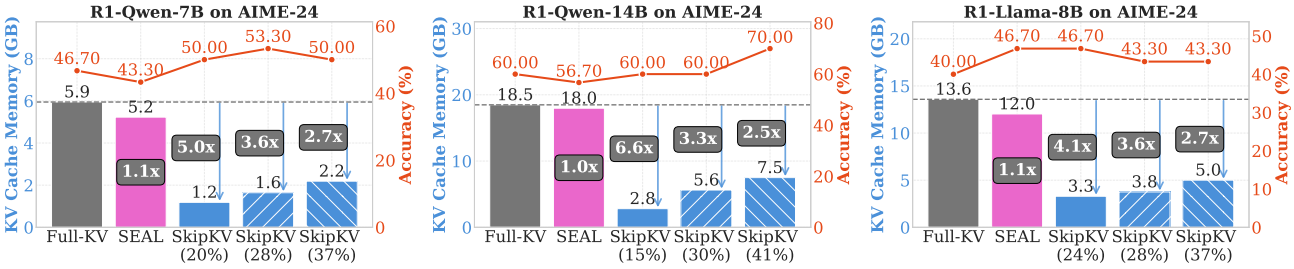

Figure 11. KV-cache memory consumption and reasoning accuracy of SkipKV under different KV budgets, compared with SEAL and Full-KV baselines on AIME-24 across multiple reasoning models. **Left**: R1-Qwen-7B; **Center**: R1-Qwen-14B; **Right**: R1-Llama-8B.

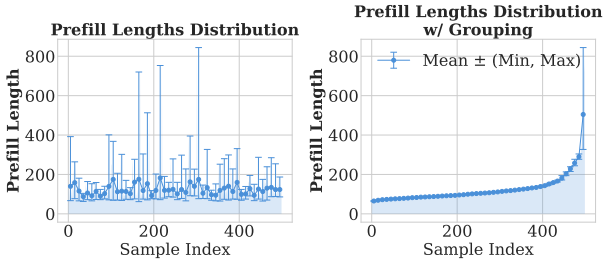

Figure 12. Visualization of the prefill token length distribution of MATH-500 and the min-max range of each 10 samples before (left) and after (right) using batch grouping.

Table 3. Effect of Batch Grouping evaluated on MATH-500 using R1-Qwen-7B with SkipKV.

KV Budget	768 (26%)	1024 (34%)	1260 (42%)	1536 (51%)
Avg. Valid Budget ↑				
w/o Batch Grouping	630 (21%)	886 (30%)	1122 (38%)	1398 (47%)
w/ Batch Grouping	**759 (25%)**	**1015 (34%)**	**1251 (42%)**	**1527 (51%)**
Accuracy (%) ↑				
w/o Batch Grouping	77.8	85.2	85.8	86.0
w/ Batch Grouping	**83.4 (+5.6)**	**86.4 (+1.2)**	**86.0 (+0.2)**	**88.0 (+2.0)**

8,192 tokens on a single A100-40GB GPU. The KV-cache budget is set to 512, under which both SkipKV and R-KV achieve comparable accuracy to FullKV. We measure the total latency and throughput (samples processed per minute) across various batch sizes, with FlashAttention-2 enabled for all methods. As shown in Table 1, we define the transition point as the batch size at which each method achieves its peak throughput—beyond which performance becomes memory-bounded. We observe that FullKV and SEAL support significantly smaller batch sizes than the eviction-based methods, resulting in earlier transition points and limited throughput scalability. In contrast, R-KV and SkipKV effectively alleviate memory pressure via their fixed-size KV caches, enabling up to 2.8× larger batch sizes and substantially higher throughput. Overall, R-KV achieves up to a 7.6× speedup over FullKV, while SkipKV further accelerates generation by 9.6× compared with FullKV. Additionally, at the same batch size, SkipKV outperforms R-KV by up to 1.7×, benefiting from its shorter generation length.

5.3 Ablation Studies

SkipKV Components. From Table 2, we observe that progressively integrating the three SkipKV components—Sentence Scoring (§ 4.1), Adaptive Steering (§ 4.2), and Batch Grouping (§ 4.3)—yields consistent improvements in both reasoning efficiency and accuracy on the AIME24 benchmark. Sentence Scoring provides moderate gains by removing redundant sentences, slightly improving accuracy and shortening the generated sequence length. Including Adaptive Steering further enhances efficiency by dynamically skipping non-execution thoughts, leading to a substantial reduction in total token length without sacrificing accuracy. Finally, combining Batch Grouping with the previous components forms the complete SkipKV configuration, which achieves the strongest overall results, improving accuracy by up to +20 % and reducing total token length by as much as 30 % across different KV budgets compared with the R-KV baseline. These results demonstrate that SkipKV's hierarchical design—progressing from local semantic filtering to global decoding coordination—effectively balances reasoning quality and efficiency in large-scale mathematical reasoning tasks.

Impact of Batch Grouping on Valid KV Budget. To evaluate the effectiveness of our batch grouping technique in multi-batch decoding, we visualize the prefill token length distribution of the MATH-500 dataset before and after applying batch grouping, and analyze its impact on the valid KV budget and overall accuracy. As shown in Fig. 12, the prefill token lengths vary significantly across the dataset, resulting in large intra-batch variation. However, after sorting and grouping, the prefill lengths increase smoothly from the first to the last sample, substantially reducing the variation within each batch. This reduction in variation decreases the amount of padding required, allowing the average valid KV budget to approach the nominal KV budget and thereby reducing the memory overhead caused by padding tokens. As summarized in Table 3, batch grouping allocates nearly the entire KV budget to valid tokens, effectively preserving accuracy under lower KV budgets.

6 Conclusions

We introduced SkipKV, a sentence-oriented KV compression framework designed to enhance reasoning efficiency while maintaining accuracy. It selectively skips KV generation and storage to yield lower memory footprint. Motivated by empirical observations on sentence-level structures in LRM outputs, SkipKV introduces a sentence-primary KV eviction policy and a sentence-type adaptive steering vector for more coherent and efficient generation. To further yield robust accuracy at multi-batch decoding with SkipKV, we presented a batch grouping strategy to improve the effective KV budget allocation. Compared to the SoTA alternative of R-KV, SkipKV yields up to $\mathbf{26.7}\%$ accuracy improvement at similar compression budget while generating up to $\mathbf{1.6}\times$ fewer tokens. Additionally, in multi-batch settings SkipKV yields a throughput improvement of up to $\mathbf{9.6}\times$ compared to the baseline FullKV LRM, at similar accuracy.

7 Discussions

Batch grouping method (§4.3) in SkipKV relies on request length characteristics and may be less effective under highly variable or extreme long-context scenarios. In such cases, residual padding overhead can still limit efficiency gains. While dynamic grouping strategies that adapt to prompt-length variability could help mitigate this issue, we leave a systematic exploration of such approaches to future work.

8 Acknowledgments

Initial ideation and majority of the work was done during Jiayi's internship at Intel. This work is co-funded by Intel Strategic Research Sectors (SRS) - Systems Integration SRS & Devices SRS. Additionally, we would like to acknowledge Akshat Ramachandran from Georgia Institute of Technology, Stefano Pellerano, Hai Li, and Arnab Raha from Intel for their insightful conversations and feedback throughout the duration of this project.

References

Ahn, J., Verma, R., Lou, R., Liu, D., Zhang, R., and Yin, W. Large language models for mathematical reasoning: Progresses and challenges. In *The 18th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 225, 2024.

Azizi, S., Potraghloo, E. B., and Pedram, M. Activation steering for chain-of-thought compression. *arXiv preprint arXiv:2507.04742*, 2025.

Behnam, P., Fu, Y., Zhao, R., Tsai, P.-A., Yu, Z., and Tumanov, A. Rocketkv: Accelerating long-context llm inference via two-stage kv cache compression. 2025.

Cai, Z., Xiao, W., Sun, H., Luo, C., Zhang, Y., Wan, K., Li, Y., Zhou, Y., Chang, L.-W., Gu, J., et al. R-kv: Redundancy-aware kv cache compression for training-free reasoning models acceleration. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Chen, R., Zhang, Z., Hong, J., Kundu, S., and Wang, Z. Seal: Steerable reasoning calibration of large language models for free. In *Second Conference on Language Modeling*, 2025a.

Chen, X., Xu, J., Liang, T., He, Z., Pang, J., Yu, D., Song, L., Liu, Q., Zhou, M., Zhang, Z., Wang, R., Tu, Z., Mi, H., and Yu, D. Do NOT think that much for 2+3=? on the overthinking of long reasoning models. In *Forty-second International Conference on Machine Learning*, 2025b.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Hu, J., Huang, W., Wang, W., Li, Z., Hu, T., Liu, Z., Chen, X., Xie, T., and Shan, Y. Raas: Reasoning-aware attention sparsity for efficient llm reasoning. 2025.

Jain, N., Han, K., Gu, A., Li, W.-D., Yan, F., Zhang, T., Wang, S., Solar-Lezama, A., Sen, K., and Stoica, I. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Kang, H., Zhang, Q., Kundu, S., Jeong, G., Liu, Z., Krishna, T., and Zhao, T. Gear: An efficient kv cache compression recipe for near-lossless generative inference of llm. *NeurIPS ENLSP Workshop*, 2024.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

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

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

Liu, G., Li, C., Ning, Z., Lin, J., Yao, Y., Ke, D., Guo, M., and Zhao, J. Freekv: Boosting kv cache retrieval for efficient llm inference. *arXiv preprint arXiv:2505.13109*, 2025a.

Liu, R., Sun, Y., Zhang, M., Bai, H., Yu, X., YU, T., Yuan, C., and Hou, L. Quantization hurts reasoning? an empirical study on quantized reasoning models. In *Second Conference on Language Modeling*, 2025b.

Liu, X., Tang, Z., Dong, P., Li, Z., Liu, Y., Li, B., Hu, X., and Chu, X. Chunkkv: Semantic-preserving kv cache compression for efficient long-context llm inference. *Advances in Neural Information Processing Systems*, 2025c.

Liu, Z., Yuan, J., Jin, H., Zhong, S., Xu, Z., Braverman, V., Chen, B., and Hu, X. Kivi: a tuning-free asymmetric 2bit quantization for kv cache. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 32332–32344, 2024.

Luo, H., Sun, Q., Xu, C., Zhao, P., Lou, J.-G., Tao, C., Geng, X., Lin, Q., Chen, S., Tang, Y., et al. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. In *The Thirteenth International Conference on Learning Representations*.

Mahbub, S., Kundu, S., and Xing, E. P. PRISM: Enhancing PRotein inverse folding through fine- grained retrieval on structure-sequence multimodal representations. In *The Fourteenth International Conference on Learning Representations*, 2026.

Mathematical Association of America. AIME 2024 Problems. https://artofproblemsolving.com/wiki/index.php/2024_AIME_I_Problems, 2024. Accessed: 2025-08-30.

Ramachandran, A., Neseem, M., Sakr, C., Venkatesan, R., Khailany, B., and Krishna, T. ThinKV: Thought-adaptive KV cache compression for efficient reasoning models. In *The Fourteenth International Conference on Learning Representations*, 2026.

Reimers, N. and Gurevych, I. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019.

Tang, J., Zhao, Y., Zhu, K., Xiao, G., Kasikci, B., and Han, S. Quest: Query-aware sparsity for efficient long-context llm inference. In *Forty-first International Conference on Machine Learning*.

Trinh, T. H., Wu, Y., Le, Q. V., He, H., and Luong, T. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

Wang, J. and Chen, Y. A review on code generation with llms: Application and evaluation. In *2023 IEEE International Conference on Medical Artificial Intelligence (MedAI)*, pp. 284–289. IEEE, 2023.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*.

Yang, C., Si, Q., Duan, Y., Zhu, Z., Zhu, C., Li, Q., Lin, Z., Cao, L., and Wang, W. Dynamic early exit in reasoning models. In *The Fourteenth International Conference on Learning Representations*, 2026.

Yang, Y., Cao, Z., Chen, Q., Qin, L., Yang, D., Zhao, H., and Chen, Z. Kvsharer: Efficient inference via layer-wise dissimilar kv cache sharing. *arXiv preprint arXiv:2410.18517*, 2024.

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

Zhong, L. and Wang, Z. Can llm replace stack overflow? a study on robustness and reliability of large language model code generation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 21841–21849, 2024.

Algorithm 1 Skipping KV Cache Storage

Require: KV cache length before eviction $l_t^{(\text{cs})}$, KV cache budget B, number of sentences n, sentence ranges in generation and cache spaces $\mathcal{M}^{(\text{gs})}$, $\mathcal{M}^{(\text{cs})}$, token span-PSS score lookup table $\mathcal{T}$.
Ensure: Compressed cache $\mathbf{K}_{\text{cache}}, \mathbf{V}_{\text{cache}}$, updated cache ranges $\mathcal{M}^{(\text{cs})}$.

 # Pre-eviction update
 # Case (1): Initialize empty cache ranges.
if $\mathcal{M}^{(\text{cs})}$ is empty **then**
  for each sentence range $\mathcal{M}_i^{(\text{gs})}$ where $e_i^{(\text{gs})} \leq l_t^{(\text{cs})}$ **do**
    Initialize $\mathcal{M}_i^{(\text{cs})}$ using Eq. (9).
  end for
  # Case (2): Append new sentence ranges.
else
  for each sentence range $\mathcal{M}_i^{(\text{gs})}$ where $i \geq n$ **do**
    Update $\mathcal{M}_i^{(\text{cs})}$ using Eq. (10).
    $\mathcal{M}^{(\text{cs})} \leftarrow \mathcal{M}^{(\text{cs})} \cup \{\mathcal{M}_i^{(\text{cs})}\}$, **if** $e_i^{(\text{cs})} \leq l_t^{(\text{cs})}$.
  end for
end if

 # KV eviction
Map redundancy scores to cache ranges $\Phi(\mathcal{T})$ via Eq. (8); evict redundant tokens and update $\mathbf{K}_{\text{cache}}, \mathbf{V}_{\text{cache}}$ with fixed budget B via Eq. (6).

 # Post-eviction update
Update $\mathcal{M}_i^{(\text{cs})}$ using Eq. (11) , $\forall \mathcal{M}_i^{(\text{cs})} \in \mathcal{M}^{(\text{cs})}$.
Update the sentence count $n \leftarrow \text{len}(\mathcal{M}^{(\text{gs})})$.

A Appendix

A.1 SkipKV: Algorithm

We provide the pseudo-code of the SkipKV storage-skipping mechanism in Algorithm 1. Suppose there are totally n sentences, we define the sets of sentence spans in the generation space and their corresponding ranges in the KV cache as

$$\mathcal{M}^{(\text{gs})} = \{(b_i^{(\text{gs})}, e_i^{(\text{gs})})\}_{i=1}^{n}, \quad \mathcal{M}^{(\text{cs})} = \{(b_i^{(\text{cs})}, e_i^{(\text{cs})})\}_{i=1}^{n}, \tag{13}$$

where the i^{th} sentence span in the original generation space (gs) should be mapped to that in the cache space (cs) as $\Phi(b_i^{(gs)}, e_i^{(gs)}) \rightarrow (b_i^{(cs)}, e_i^{(cs)})$ using the cache range monitoring logic mentioned in §4.1. At each eviction step, we obtain the compressed $\mathbf{K}_{\text{cache}}, \mathbf{V}_{\text{cache}}$, and the updated sentence ranges in cache space $\mathcal{M}^{(\text{cs})}$ with Algorithm 1.

The complete SkipKV procedure including both KV storage skipping and KV generation skipping is detailed in Algorithm 2. For clarity, we define the **newline delimiter set**,

Algorithm 2 SkipKV Algorithm

Require: Large Reasoning Model $f(\theta, \cdot)$, input content X, KV cache budget B, compression step interval Δ_t, newline delimiter set $\mathcal{D}$, non-executable keyword set $\mathcal{N}$, steering layer index L_s, steering increment factor γ, initial steering strength α_0, steering vector $\mathbf{v}$, maximum generation length N
Ensure: Generated text Y.

while $t < N$ **do**
  # 1 ∼ 2. Record labeled generated sentence ranges
  $i \leftarrow 0, b_i^{(gs)} \leftarrow 0$
  for x_t in X **do**
    if $x_t \in \mathcal{D}$ **then**
      $\mathcal{M}_i^{(gs)} \leftarrow (b_i^{(gs)}, x_t)$, $b_{i+1}^{(gs)} \leftarrow x_t + 1, i \leftarrow i + 1$
      if $(\forall x_t \in \mathcal{M}_i^{(gs)}) \in \mathcal{N}$ **then**
        Label $\mathcal{M}_i^{(gs)}$ with 'Non-exe'
      end if
    end if
  end for

  # 3. Skip KV Storage
  $\mathbf{K}_{\text{cache}}, \mathbf{V}_{\text{cache}}, y \leftarrow f(\theta, X)$
  for k in decoder layers L **do**
    if $t \bmod \Delta_t == 0$ **then**
      Update $\mathbf{K}_{\text{cache}}, \mathbf{V}_{\text{cache}}$, and $\mathcal{M}^{(\text{cs})}$ using Alg. 1
    end if
    # Skip KV Generation
    $\mathbf{H}_k \leftarrow \mathbf{H}_k + \alpha_t \cdot \mathbf{v}$, **if** $k == L_s \wedge x_t \in \mathcal{D}$
  end for

  # 4 ∼ 5. Compute PSS
  Compute redundancy scores for sentences in $\mathcal{M}^{(\text{gs})}$ and record redundant set $\mathcal{T}$ using Eq. (7).

  # Update Steering Strength
  $N_o \leftarrow \text{count}(\mathcal{M}_i^{(\text{gs})}.\text{key}() == \text{'Non-exe'}, \forall i)$
  $\alpha_t \leftarrow \alpha_0 + N_o \cdot \gamma$

  # Auto-regressive update
  $y \rightarrow Y; \quad y \rightarrow X; \quad t \leftarrow t + 1$
end while

which specifies where sentence ranges are monitored and steering vectors are applied, as follows:

$$\mathcal{D} = \{\text{"\textbackslash n"}, \text{" .\textbackslash n"}, \text{")\textbackslash n"}, \text{"\textbackslash n\textbackslash n"}, \text{" .\textbackslash n\textbackslash n"}, \text{")\textbackslash n\textbackslash n"}\}.$$

To improve robustness, we merge consecutive delimiter tokens to avoid over-fragmentation during sentence segmentation. Additionally, for mixed contexts, the delimiter set can be empirically customized based on the generation domain. In our experiments on LiveCodeBench, we find that the proposed delimiter set remains effective. Importantly, we

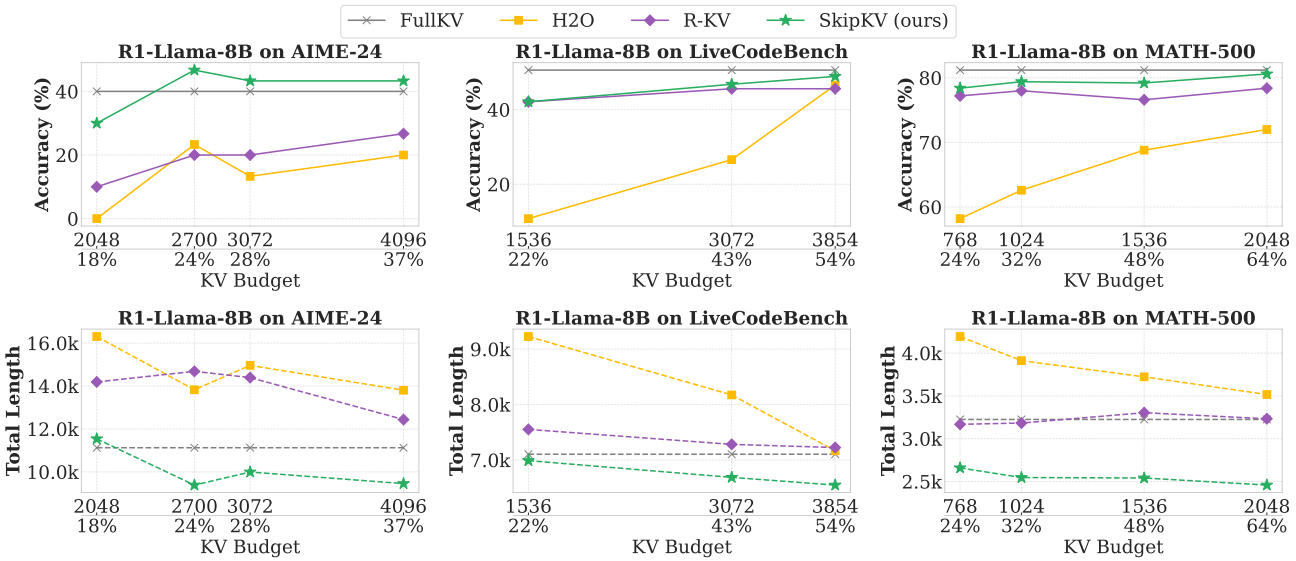

Figure 13. Comparison of accuracy and total generation token length on SkipKV under different KV budget with H2O, R-KV, and FullKV across three datasets and R1-Llama-8B model.

exclude syntax-critical delimiters (e.g., `:\n`) that commonly appear in control-flow constructs such as `if-else` or `for` loops, as splitting on these could fragment semantically coherent code blocks and increase the risk of premature eviction.

The **non-executable keyword set**, which determines where the steering strength is increased, is defined as:

$$\mathcal{N} = \{\texttt{"Wait"}, \texttt{"Alternatively"}, \texttt{"again"}\}.$$

These sets are initialized once before decoding and referenced to record the labeled input sentence ranges.

A.2 Experimental Setup

Hyper-parameters. Following R-KV, we compress and update KV cache every 128 decoding steps and set the attention-redundancy score trade-off factor to $\sigma = 0.1$. The similarity threshold τ in the sentence-scoring metric is selected within the range of 0.95 to 0.99. The steering strength α is initialized as 1 or 1.25, and the steering strength increment factor γ is set to 0.02. We insert the steering vector in the 20-th layer of R1-Qwen-7B and R1-Llama-8B, and in the 35-th layer of R1-Qwen-14B.

Baselines. We compare our method against eviction-based KV compression methods H2O (Zhang et al., 2023), R-KV (Cai et al., 2025), and the steering-based efficient reasoning method SEAL (Chen et al., 2025a). FullKV is included as a reference method that keeps the full KV cache, providing the gold standard for decoding accuracy.

Table 4. Comparison of latency and accuracy on GSM8K under different batch sizes using vLLM.

Metric	Batch-size	64	32	16	8
Total Latency (min) ↓	R-KV	51	55	72	117
	SkipKV (ours)	**44**	**52**	**68**	**112**
Accuracy (%) ↑	R-KV	73.8	81.7	85.3	87.1
	SkipKV (ours)	**84.4**	**86.8**	**88.2**	**88.6**

A.3 Additional Experimental Results

Evaluation on the Llama Model Family. Fig. 13 shows the accuracy and total token length on R1-Llama-8B for MATH-500, AIME-24 and LiveCodeBench datasets comparing SkipKV with prior eviction strategies including H2O and R-KV and the FullKV baseline. Across all three datasets, SkipKV consistently achieves the best trade-off between accuracy and generation efficiency. In terms of accuracy, SkipKV either outperforms or matches FullKV across all KV budgets, while significantly outperforming H2O and R-KV, especially under tighter KV constraints. In terms of efficiency, SkipKV consistently produces the shortest total generation length across all settings, indicating more concise reasoning. Notably, it reduces token length substantially compared to FullKV while simultaneously improving or preserving accuracy, demonstrating that redundant reasoning can be effectively removed without harming performance. These results extends our evaluation beyond Qwen models in the main content, demonstrating that SkipKV generalizes effectively to other model families such as Llama.

System-Level Evaluation on vLLM. To evaluate system-level performance, we further integrate SkipKV into the vLLM v1 framework (Kwon et al., 2023), building upon the R-KV implementation. Unlike the HuggingFace-based

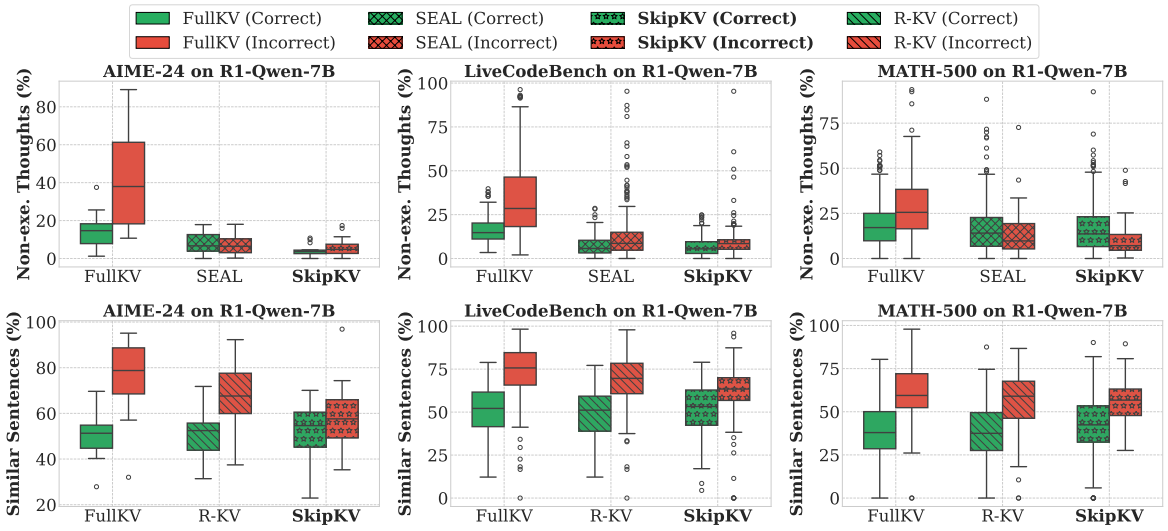

Figure 14. Comparison of the ratio of non-execution thoughts (top) and high-similarity sentences (bottom) generated by different methods on AIME-24, LiveCodeBench, and MATH-500 using R1-Qwen-7B. The boxplots show distributions for samples that were answered correctly (green) and incorrectly (red) of each method.

PyTorch setup, which uses static batching, vLLM adopts continuous batching with a paged KV cache, eliminating explicit prefill padding within each batch. Consequently, we do not apply the batch-grouping optimization in this setting and instead focus on evaluating the core component of SkipKV—sentence-aware KV eviction and selection—under realistic inference-time scheduling and memory management.

As shown in Tab. 4, under a fixed KV budget (B=512), R-KV exhibits noticeable accuracy degradation as batch size increases, indicating limited robustness under multi-request decoding. In contrast, SkipKV maintains stable accuracy across different batch sizes and achieves strong performance even at larger batch sizes, while also reducing end-to-end latency. These results demonstrate that SkipKV is compatible with vLLM's paged KV cache and continuous batching mechanism, and can effectively improve both efficiency and accuracy in practical multi-request inference scenarios.

A.4 Analysis of Sentence-level Properties after Eviction

Fig. 14 compares the reasoning behavior of different KV eviction strategies on AIME-24, LiveCodeBench, and MATH-500 using R1-Qwen-7B. We set the KV budget to 2220 (20%), 2000 (28%), 1024 (30%) for both R-KV and SkipKV on each dataset, under which SkipKV attains the same accuracy as FullKV. The top row of the figure illustrates the fraction of non-execution thoughts, reflecting the frequency of unnecessary re-validation steps, while the bottom row reports the proportion of high-similarity sentences, which indicate repetitive reasoning patterns.

As discussed in Observations 4 and 5 in §4, comparing correct and incorrect samples reveals that non-executable and redundant reasoning accumulates more heavily in incorrect generations. SkipKV effectively narrows this gap, reducing unnecessary and repetitive thoughts in both cases. Quantitatively, SkipKV generates approximately $4\times$ and $8\times$ fewer non-execution thoughts in correct and incorrect generations compared to FullKV, and further outperforms SEAL by $1.8\times$ and $1.3\times$, respectively.

Moreover, SkipKV reduces the variance of the non-execution-thought ratio across samples, enabled by its sample-wise adaptive steering mechanism. Empirically, applying an initial steering strength $\alpha_0 = 1.0$ reduces the number of non-execution thoughts from a wide range (up to $\sim$1000) to a controlled range (up to $\sim$50). With $\gamma = 0.02$, the resulting $\alpha_t \in [1, 2]$ remains within a safe regime identified in prior work (Chen et al., 2025a), while enabling dynamic, sample-dependent adjustment based on the evolving generation.

We also observed that, although R-KV aims to remove redundant tokens, it only reduces similar sentences in incorrect answers by up to 11%, whereas SkipKV—explicitly designed to target redundant sentences—consistently achieves over $2\times$ greater reduction. Overall, SkipKV demonstrates a clear advantage in maintaining concise, execution-oriented reasoning under constrained KV budgets.

A.5 Empirical Study of Generated Outputs

In this section, we present qualitative examples from the MATH-500 dataset to compare the generation behaviors of R-KV and SkipKV (Figures 15 and 16). We visualize

the generated outputs of the R1-Qwen-7B reasoning model under a KV-cache budget of 1024. In the selected example, both methods produce the correct final answer; however, SkipKV generates approximately 20% fewer tokens than R-KV. In both figures, non-execution thoughts are highlighted in yellow and final-answer segments are highlighted in a blue box. The darkness of the red text indicates the number of attention heads selecting each token. From the visualization of eviction patterns, SkipKV primarily removes complete sentences from the KV cache, whereas R-KV tends to evict fragmented tokens. The redundancy-based scoring of R-KV often leads it to remove numerical tokens that lie along crucial mathematical reasoning paths, disrupting logical consistency and resulting in longer generations. Notably, R-KV also evicts tokens from the final-answer region, which confuses the model and triggers unnecessary re-validation steps. In contrast, SkipKV preserves the key reasoning and answer segments, demonstrating more coherent and efficient generation behavior.

<|User|>Answer the following questions. You should think step-by-step and put your final answer within \boxed{}.
Question: Expand and simplify completely: \begin{align*}
x\left(x(1+x)+2x\right)-3(x^2-x+2)
\end{align*}<|Assistant|>
[Re-validate for 5 times and generate 2125 tokens in total]
Okay, so I have this algebra problem to solve: expand and simplify completely the expression \(x\left(x(1+x)+2x\right)-3(x^2-x+2) \). Hmm, let me try to figure this out step by step. I remember that expanding means I need to distribute the multiplication over addition or subtraction. So, I should probably start by looking at each part of the expression separately.
First, let me look at the first part: \(x\left(x(1+x)+2x\right) \). I see that inside the parentheses, there are two terms: \(x(1+x) \) and \(2x \). I think I need to simplify inside the parentheses first before multiplying by the x outside. So, let me work on \(x(1+x) \).
Multiplying x by (1 + x) should be straightforward. Using the distributive property, that would be \(x*1 + x*x \), which is \(x + x^2 \). Okay, so that simplifies the first part. Now, the expression inside the parentheses becomes \(x + x^2 + 2x \). Wait, I have \(x + x^2 + 2x \). I can combine like terms here. The x and 2x are like terms, so adding them together gives me \(3x \). So, now the expression inside the parentheses is \(x^2 + 3x \).
So, the first part simplifies to \(x(x^2 + 3x) \). Now, I need to distribute the x across both terms inside the parentheses. Let's do that. Multiplying x by \(x^2 \) gives me \(x^3 \), and multiplying x by 3x gives me \(3x^2 \). So, the first part becomes \(x^3 + 3x^2 \). Got that down.
Now, moving on to the second part of the original expression: \(-3(x^2 - x + 2) \). I need to distribute the -3 across each term inside the parentheses. Let's see, multiplying -3 by \(x^2 \) gives me \(-3x^2 \), multiplying -3 by (-x) gives me \(+3x \) (because a negative times a negative is a positive), and multiplying -3 by 2 gives me \(-6 \). So, putting that all together, the second part simplifies to \(-3x^2 + 3x - 6 \).
Now, I have both parts simplified: the first part is \(x^3 + 3x^2 \) and the second part is \(-3x^2 + 3x - 6 \). The next step is to combine these two results. So, let me write them out together:
\(x^3 + 3x^2 - 3x^2 + 3x - 6 \).
Now, I need to combine like terms. Let's look for terms with the same power of x. The \(3x^2 \) and \(-3x^2 \) are like terms because they both have \(x^2 \). Adding those together: \(3x^2 - 3x^2 = 0 \). So, those cancel each other out. That leaves me with \(x^3 + 3x - 6 \).

Wait, so after combining like terms, the expression simplifies to \(x^3 + 3x - 6 \). Let me double-check my work to make sure I didn't make any mistakes.

Starting again, the original expression is \(x(x(1+x)+2x) - 3(x^2 - x + 2) \).
First, inside the first parentheses: \(x(1+x) + 2x \). That's \(x + x^2 + 2x \), which is \(x^2 + 3x \).
Then, multiplying by x: \(x(x^2 + 3x) = x^3 + 3x^2 \). That seems right.
Then, the second part: \(-3(x^2 - x + 2) \). Distribute the -3: \(-3x^2 + 3x - 6 \). That also looks correct.
Combining both parts: \(x^3 + 3x^2 - 3x^2 + 3x - 6 \). The \(3x^2 \) and \(-3x^2 \) do indeed cancel each other, leaving \(x^3 + 3x - 6 \).

Hmm, is there any other term I might have missed? Let me check each step again.
1. Expanded \(x(1+x) \) to \(x + x^2 \). Correct.
2. Added \(2x \) to get \(x + x^2 + 2x = x^2 + 3x \). Correct.
3. Distributed the outer x: \(x(x^2 + 3x) = x^3 + 3x^2 \). Correct.
4. Distributed the -3: \(-3x^2 + 3x - 6 \). Correct.
5. Combined all terms: \(x^3 + 3x^2 - 3x^2 + 3x - 6 \). Correct.
6. Simplified: \(x^3 + 0x^2 + 3x - 6 \), which is \(x^3 + 3x - 6 \). Correct.

I don't think I missed anything. So, the simplified expression is \(x^3 + 3x - 6 \).

Just to be thorough, let me plug in a value for x and see if the original expression and the simplified one give the same result. Let's choose x = 1.
Original expression: \(1*(1*(1+1) + 2*1) - 3*(1^2 - 1 + 2) \).
...
Now, let's compute the simplified expression at x = 1: \(1^3 + 3*1 - 6 = 1 + 3 - 6 = -2 \). Same result. Good.
Let me try another value, say x = 2.
Original expression: \(2*(2*(2+1) + 2*2) - 3*(4 - 2 + 2) \).
...
Simplified expression: \(2^3 + 3*2 - 6 = 8 + 6 - 6 = 8 \). Same result. Good.
One more test, maybe x = 0.
...
So, after expanding and simplifying, the expression is \(x^3 + 3x - 6 \). I think that's the correct answer.
Final Answer
The simplified expression is \boxed{x^3 + 3x - 6}.

Figure 15. Qualitative example of **R-KV** responses on the MATH-500 dataset. The darkness of red denotes how many heads select the token. Non-execution sentences starting re-validation are highlighted in yellow, where each is followed by few execution thoughts, and the answers are highlighted in a blue box. R-KV frequently selects fragmented tokens within execution reasoning and always includes parts of the answer itself.

<|User|>Answer the following questions. You should think step-by-step and put your final answer within \boxed{}.
Question: Expand and simplify completely: \begin{align*}
x\left(x(1+x)+2x\right)-3(x^2-x+2)
\end{align*}<|Assistant|>
[Re-validate for 3 times and generate 1742 tokens in total]
Okay, so I have this algebra problem to solve: expand and simplify completely the expression \(x\left(x(1+x)+2x\right)-3(x^2-x+2) \). Hmm, let me try to figure this out step by step. I remember that expanding means I need to distribute the multiplication over addition or subtraction. So, I should probably start by looking at each part of the expression separately.
First, let me look at the first part: \(x\left(x(1+x)+2x\right) \). I see that inside the parentheses, there are two terms: \(x(1+x) \) and \(2x \). I think I need to simplify inside the parentheses first before I multiply by the x outside. So, let me work on \(x(1+x) \).
Multiplying x by (1 + x) should be straightforward. Using the distributive property, that would be \(x*1 + x*x \), which is \(x + x^2 \). So, replacing that back into the expression, the first part becomes \(x\left(x + x^2 + 2x\right) \). Wait, hold on, I have \(x(1+x) \) which is \(x + x^2 \), and then I add \(2x \) to that. So, combining like terms inside the parentheses: \(x + 2x \) is \(3x \). So, the expression inside the parentheses simplifies to \(x^2 + 3x \).
Now, the first part is \(x(x^2 + 3x) \). I need to distribute the x across both terms inside the parentheses. So, \(x*x^2 \) is \(x^3 \), and \(x*3x \) is \(3x^2 \). So, after distributing, the first part becomes \(x^3 + 3x^2 \). Okay, that's the first part done.
Now, moving on to the second part of the original expression: \(-3(x^2 - x + 2) \). I need to distribute the -3 across each term inside the parentheses. Let's do that term by term.
First, \(-3*x^2 \) is \(-3x^2 \). Then, \(-3*(-x) \) is \(+3x \). Finally, \(-3*2 \) is \(-6 \). So, putting it all together, the second part simplifies to \(-3x^2 + 3x - 6 \).
Now, combining both parts of the original expression: the first part was \(x^3 + 3x^2 \) and the second part is \(-3x^2 + 3x - 6 \). So, putting them together, the entire expression becomes \(x^3 + 3x^2 - 3x^2 + 3x - 6 \).
Now, I need to combine like terms. Let's see: the \(x^3 \) term is alone, so that stays as is. Then, \(3x^2 - 3x^2 \) cancels out, right? Because 3x^2 minus 3x^2 is zero. So, those terms disappear. Then, we have the \(3x \) term and the constant term \(-6 \).
So, after combining like terms, the expression simplifies to \(x^3 + 3x - 6 \). Hmm, let me double-check to make sure I didn't make any mistakes.
Starting again, the original expression: \(x(x(1+x)+2x) -3(x^2 -x +2) \).
First, inside the first parentheses: \(x(1+x) + 2x \). That's \(x + x^2 + 2x \), which is \(x^2 + 3x \).
Then, multiplying by x: \(x^3 + 3x^2 \).
Second part: \(-3(x^2 -x +2) \) is \(-3x^2 + 3x -6 \).
Adding both parts: \(x^3 + 3x^2 -3x^2 + 3x -6 \).
Simplify: \(x^3 + (3x^2 -3x^2) + 3x -6 \) which is \(x^3 + 0 + 3x -6 \), so \(x^3 + 3x -6 \).
Looks like I did it right. So, the simplified expression is \(x^3 + 3x -6 \).
I don't think there are any more like terms to combine, so that should be the final simplified form.
Just to make sure, let me plug in a value for x and see if both the original expression and the simplified one give the same result. Let's choose x = 1.
Original expression: \(1*(1*(1+1) + 2*1) -3*(1^2 -1 +2) \).
Compute step by step:
First, inside the first parentheses: \(1*(1+1) = 1*2 = 2 \). Then, add 2*1 = 2. So, 2 + 2 = 4.
Multiply by 1: 1*4 = 4.
Now, the second part: \(-3*(1 -1 +2) = -3*(2) = -6 \).
So, total expression: 4 -6 = -2.
Simplified expression: \(1^3 + 3*1 -6 = 1 + 3 -6 = -2 \).
Same result. Let me try another value, say x = 2.
Original expression: \(2*(2*(2+1) + 2*2) -3*(4 -2 +2) \).
Compute step by step:
First, inside the first parentheses: \(2*(3) = 6 \). Then, add 2*2 = 4. So, 6 + 4 = 10.
Multiply by 2: 2*10 = 20.
Second part: \(-3*(4 -2 +2) = -3*(4) = -12 \).
Total expression: 20 -12 = 8.
Simplified expression: \(2^3 + 3*2 -6 = 8 +6 -6 = 8 \).
Same result again. Okay, so that seems to confirm that my simplification is correct.
So, after expanding and simplifying, the expression becomes \(x^3 + 3x -6 \).
Final Answer
The simplified expression is \boxed{x^3 + 3x - 6}.

Figure 16. Qualitative example of **SkipKV** responses on the MATH-500 dataset. SkipKV primarily evicts entire sentences instead of fragmented tokens, avoiding interrupting the reasoning path.