LazyEviction: Lagged KV Eviction with Attention Pattern Observation for Efficient Long Reasoning

Haoyue Zhang[1], Hualei Zhang[1], Xiaosong Ma[2], Jie Zhang[1]*, Song Guo[1]
[1]The Hong Kong University of Science and Technology
[2]The Hong Kong Polytechnic University
{hzhangex, hzhangfz}@connect.ust.hk, xiaosong16.ma@connect.polyu.hk
{csejzhang@, songguo@cse}.ust.hk

Abstract

Large Language Models (LLMs) exhibit enhanced capabilities by Chain-of-Thought reasoning. However, the extended reasoning sequences introduce significant GPU memory overhead due to increased key-value (KV) cache. Existing KV cache compression methods mitigate memory bottlenecks but struggle in long reasoning tasks. In this paper, we analyze attention patterns in reasoning tasks and reveal a **Token Importance Recurrence** phenomenon: a large proportion of tokens regain high attention after multiple decoding steps, which is failed to capture by existing works and may lead to unpredictable eviction on such periodically critical tokens. To address this, we propose **LazyEviction**, an observation window-based lagged eviction framework retaining latent recurring tokens by an eviction policy informed by token recurrence patterns. Extensive experiments demonstrate that LazyEviction reduces KV cache by 50%~70% while maintaining comparable accuracy, outperforming existing KV cache compression baselines. Our implementation code can be found at https://github.com/Halo-949/LazyEviction.

1 Introduction

Large Language Models (LLMs) have emerged advanced capabilities in complex reasoning tasks by enabling Chain-of-Thought (CoT) to elicit step-by-step inference (Wei et al., 2022; Kojima et al., 2022; Feng et al., 2023). Recent advancements, such as OpenAI's o1 (OpenAI et al., 2024) and DeepSeekR1 (Guo et al., 2025), further demonstrate that scaling up CoT lengths from hundreds to thousands of reasoning steps could continuously improve LLM reasoning. However, the increased reasoning sequences introduce substantial memory and computational overhead. Due to the autoregressive nature of LLM decoding, longer CoT lead to proportional increases in GPU memory for caching key-value (KV) states. These issues become particularly pronounced when reasoning sequences extend into thousands of reasoning tokens, resulting in significant GPU memory costs. For example, in mathematics (Cobbe et al., 2021; Hendrycks et al., 2021) and programming tasks (Chen et al., 2021), the reasoning sequences can grow up to 16k tokens, resulting in more than 100GB KV cache with a batch size of 32, which exceeds the memory capacity of even high-end GPUs.

KV cache compression has emerged as a promising approach to alleviate the memory bottleneck caused by increasing KV cache sizes. Unlike traditional long-context input tasks that focus on the prefilling phase (Li et al., 2024; Cai et al., 2024; Feng et al., 2024; Fu et al., 2024), long-reasoning tasks perform long-context generation that requires compression in multiple decoding steps. The critical challenge lies in compressing generated KV caches while minimizing performance degradation. Existing approaches mitigate this through selective eviction strategies: early work (Xiao et al., 2023) preserves subsets of recent and initial KV pairs, while others (Oren et al., 2024; Chen et al., 2024; He et al., 2025; Zhang et al., 2023; Adnan et al., 2024) leverage attention scores to prioritize critical tokens and evict others, which can be further divided into two types: (a) Current Attention-based Eviction (Oren et al., 2024; Chen et al., 2024; He et al., 2025) and (b) Cumulative Attention based Eviction (Zhang et al., 2023; Adnan et al., 2024; Ghadia et al., 2025; Hu et al., 2025). However, as shown in Fig. 1, these works are greedy eviction strategies at each decoding step, which discard tokens deemed temporarily unimportant while overlooking their potential future importance.

We empirically analyze attention patterns during multi-step reasoning processes. Our investigation reveals a frequent **Token Importance Recurrence (TIR)** phenomenon: a large proportion of tokens in reasoning tasks receive renewed attention after

* Corresponding author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 36335–36352

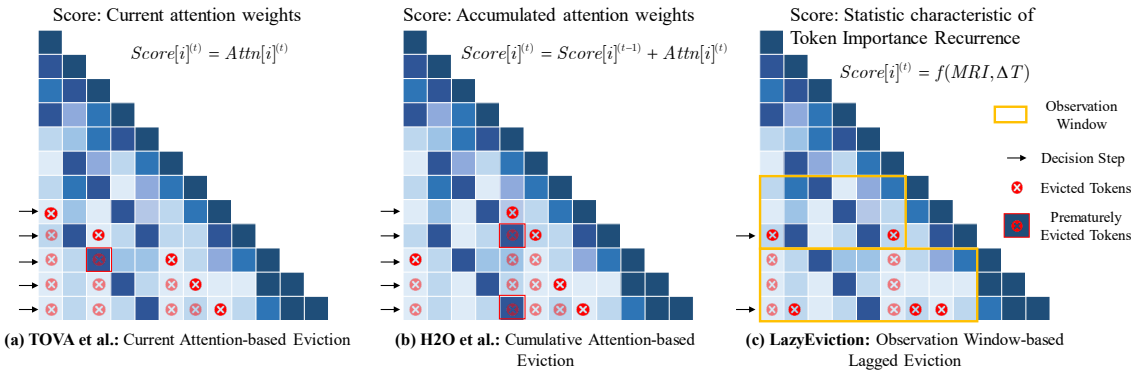


Figure 1: Comparison of different KV Eviction methods. The dark squares represent that the token has a higher attention score. (a) Current Attention-based Eviction executes stepwise evictions using immediate attention scores. (b) Cumulative Attention-based Eviction integrates historical attention for eviction decisions. Both (a) and (b) fail to preserve recurring tokens during their low-attention intervals. (c) LazyEviction performs lagged KV evictions based on the observation window to detect latent recurring tokens and prevent prematurely discarding them.

multiple decoding steps. In this paper, we refer to such tokens as **recurring tokens**, where the attention score of recurring tokens may be low within an interval while suddenly increasing in the future. Empirical analysis demonstrates that these tokens typically involve conditional or summary information within reasoning chains, aligning with the capabilities in verification, backtracking, and summarization in reasoning models (Gandhi et al., 2025). The recurring tokens often contain information vital for later reasoning stages, and premature eviction of recurring tokens leads to catastrophic performance degradation. Crucially, existing KV compression methods usually fail to identify such tokens due to inherent limitations, thus risking prematurely discarding crucial tokens during reasoning steps, ultimately sacrificing task accuracy.

To address this challenge, we propose LazyEviction, a novel **Observation Window-based Lagged Eviction Mechanism**. Instead of per-step greedy decisions, we perform lagged KV evictions at every W decoding steps to detect latent recurring tokens and prevent prematurely discarding them. There are two key components: (1) **Recurrence Interval Tracking:** we record each token's Maximum Recurrence Interval (MRI) (i.e., the longest observed period between its attention spikes) to capture the temporal variation of token importance. (2) **MRI-Centric Eviction Policy:** we prioritize evicting tokens whose ΔT exceeds their MRI, where ΔT is the time elapsed since the token's last important recurrence. Tokens with $\Delta T < MRI$ are retained as they still have the potential to become important in the future. LazyEviction shifts from step-wised greedy eviction to window-wised predictive retention: by continuously tracking MRI and executing lagged eviction during observation windows, we can effectively detect latent recurring tokens and prevent prematurely discarding them. To sum up, our key contributions are:

- We observe token importance recurrence in the reasoning task, where specific tokens exhibit recurrent attention patterns. These recurring tokens often represent foundational formulations or intermediate conclusions that are crucial for maintaining knowledge continuity in multi-step reasoning.
- We propose LazyEviction, an observation window-based lagged KV eviction scheme integrating recurrence interval tracking. Our framework enables prediction of future token importance, thereby preventing premature eviction of recurring tokens.
- We have implemented LazyEviction on the latest large reasoning models ranging from 4B to 32B. It has been verified on multiple reasoning datasets that our method can maintain comparable performance while reducing 50%~70% KV budget, surpassing the SOTA KV cache compression methods.

2 Related Works

Various approaches have been developed to improve LLM's efficiency in handling long reasoning tasks, which can be classified into three main categories: KV cache compression, reasoning path compression, and system-level optimizations.

KV cache compression. Considering the inherent sparsity of attention mechanisms, these works fo-

cus on retaining critical KV pairs while evicting non-essential ones within constrained memory budgets. We summarize the existing works as follows:

(1) *Static-Position retention*: StreamingLLM (Xiao et al., 2023) employs static retention of initial and recent tokens under fixed budgets. While sustaining text coherence, their rigid retention policies cannot detect latent recurring tokens.

(2) *Current Attention-based Eviction*: TOVA (Oren et al., 2024), NACL (Chen et al., 2024), and TreeKV (He et al., 2025) execute stepwise evictions using immediate attention scores. This myopic strategy fails to preserve recurring tokens during low-attention intervals.

(3) *Cumulative Attention-based Eviction*: Scissorhands (Liu et al., 2023), H2O (Zhang et al., 2023), Keyformer (Adnan et al., 2024), and MorphKV (Ghadia et al., 2025) integrate tokens' historical attention values for eviction decisions. RaaS (Hu et al., 2025) extends this through dynamic updated timestamp. These approaches still mistakenly regard latent recurring tokens as unimportant due to continuous low historical scores.

Recent works R-KV (Cai et al., 2025) and RPC (Song et al., 2025) utilize token similarity to identify and evict redundant tokens in long reasoning tasks. Instead, LazyEviction directly focuses on the attention patterns in reasoning path. Our goal is to use the compressed KV cache to closely approximate the full KV attention map, ensuring the theoretical interpretability of performance.

Long Reasoning Compression. Recent works have focused on compressing the output of LLMs. For instance, TALE (Han et al., 2024) and SoT (Aytes et al., 2025) adopt prompt engineering to control the length of CoT outputs. TH2T (Liu et al., 2025) reduces the length of CoT by mitigating overthinking. TokenSkip (Xia et al., 2025) achieves output compression through fine-tuning on condensed CoT data. InftyThink (Yan et al., 2025) and LightThinker (Zhang et al., 2025) equip models with summarization capabilities to reduce tokens during the reasoning. Distinctively, LazyEviction maintains the original output length while enhancing memory and computational efficiency through optimized KV cache usage, demonstrating compatibility with these compression approaches.

System-level Optimizations. By optimizing GPU memory usage and batch processing, vLLM (Kwon et al., 2023) and FlashDecoding (Hong et al., 2023) imporve both time and memory efficiency in decoding. Additionally, QUEST (Tang et al., 2024) and OmniKV (Hao et al., 2025) store KV caches in CPU and dynamically schedule them during the decoding phase. Our method is based on an eviction strategy that prioritizes the importance of tokens in the future, and it is parallel and compatible with the above methods.

3 Observations and Motivations

The goal of KV cache eviction is to identify and evict KV pairs that are non-critical for future decoding steps. To preserve the correctness of subsequent decoding results, it is critical to retain tokens with potential future importance. In this section, we first analyze the previous works and observe attention patterns in reasoning tasks.

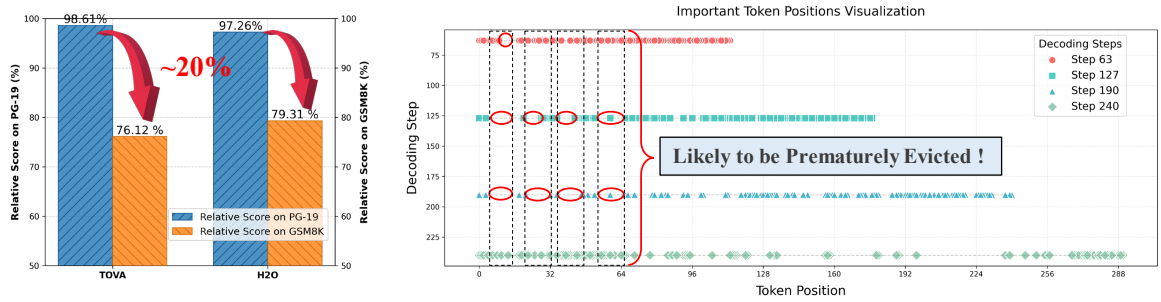
> **Finding 1** *Exiting works fail to reserve the potentially important tokens during decoding steps, thus exhibiting degraded performance in long reasoning tasks.*

We compare the performance of H2O (Zhang et al., 2023) and TOVA (Oren et al., 2024) on reasoning tasks versus general long-context generation tasks in Fig. 2(a), H2O and TOVA report competitive performance on standard language modeling dataset, but exhibit significant performance degradation on GSM8K under the same KV cache compression ratio. To diagnose this failure, we track the positions of top-50% important tokens across decoding steps in Fig. 2(b) and observe that tokens critical in later steps are often missing in earlier steps. Due to their low importance scores in intermediate steps, these tokens are prone to premature eviction in prior works.

> **Finding 2** *Most tokens (>95%) exhibit recurrent attention patterns in multi-step reasoning tasks. Since they contain critical information for the reasoning, prematurely evicting may cause knowledge incontinuity.*

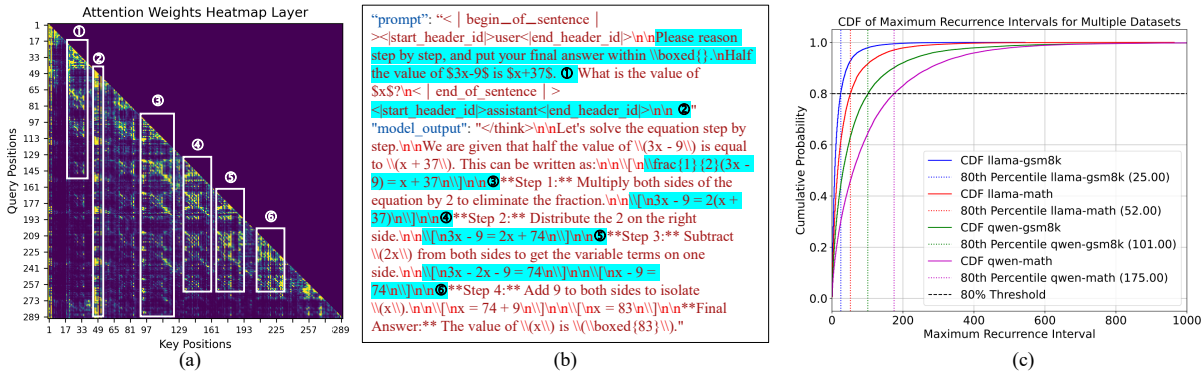
We observe a prevalent phenomenon in reasoning tasks, where tokens initially exhibit strong attention weights, receive low attention weights during later decoding steps, and subsequently regain attention after a certain decoding step interval, as illustrated in Fig. 3(a), ①–⑥. We define this phenomenon as Token Importance Recurrence (TIR).

We select a sample from the MATH500 dataset for visualization. We annotate the recurring tokens



(a) (b)

Figure 2: (a) shows the performance degradation for SOTA Methods on reasoning tasks. With the same KV cache compression ratio (e.g., 50%), compared with traditional language modeling tasks dataset PG-19 (Rae et al., 2019), the performance of both H2O and TOVA has decreased by ~20% on GSM8K dataset. (b) is the visualization of the importance variation by selecting Top-50% important tokens. Tokens at the same position show different importance at different decoding steps.



(a) (b) (c)

Figure 3: Visualization of TIR. We observe attention maps across different heads of DeepSeek-R1-Distill-Llama-8B. We find most tokens(>95%) show TIR pattern. (a) and (b) show an attention map with recurring tokens and their corresponding positions. (c) statistically analyzed the MRI distribution in different models among different tasks.

①–⑥ from Fig. 3(a) in both the input prompt and output CoT shown in Fig. 3(b). These recurring tokens contain critical information required for reasoning steps, such as the initial problem conditions (tokens ①) in the prompt, which are repeatedly referenced in subsequent reasoning steps. Additionally, intermediate results (tokens ③–⑥) generated during reasoning are reactivated in later steps.

To further validate the generality of this phenomenon, we measured the Maximum Recurrence Interval (MRI) — the longest period between consecutive re-activations of a token — across multiple reasoning models on the GSM8K and MATH500 datasets. Results show that over 95% of tokens exhibit importance recurrence (MRI > 1), confirming its ubiquity in reasoning tasks (Fig. 3(c)).

Finding 3 *Most recurring tokens' MRI is far smaller than the output length and can be detected via an observation window.*

As shown in Fig. 3(c), we statistically analyzed the MRI distribution in different models among different tasks. As the output length of the model increases, the MRI gradually becomes larger, but most tokens' MRI remian relatively small. For example, the output length of Qwen model on the MATH500 dataset can reach 8k, but we have statistically found that 80% tokens' MRI are less than 175. Therefore, as long as the window size exceeds the MRI of most tokens, the majority of recurring tokens will be identified. This finding motivates our design of the observation window-based lagged eviction mechanism.

4 LazyEviction

In this section, we introduce our LazyEviction, a novel framework of **Observation Window-based Lagged Eviction**. We first model LazyEviction as a multi-step dynamic optimization problem in Appendix D. Unlike existing works that perform KV eviction decisions at every decoding step, our

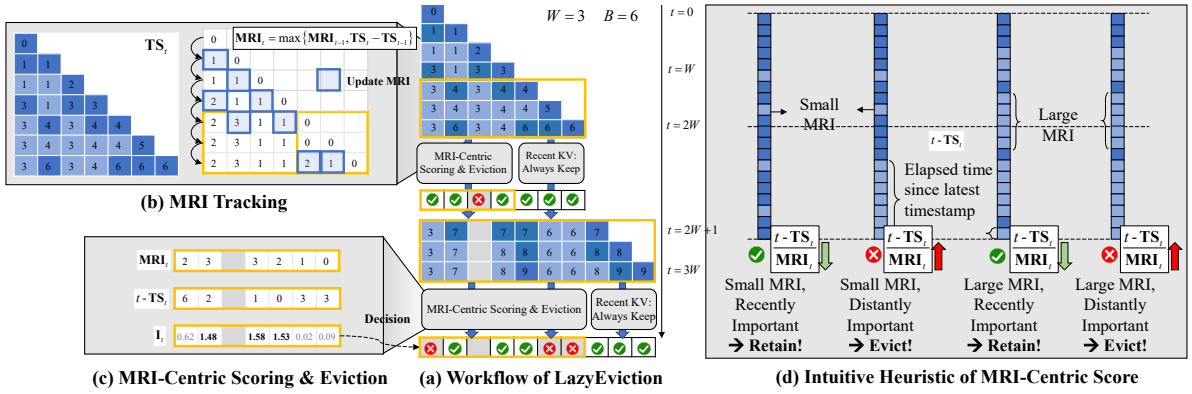


Figure 4: Overview of our proposed LazyEviction Framework, where the dark squares represent that the token has a relatively higher attention score. (a) LazyEviction performs eviction decisions at intervals of W steps. The workflow contains two key operations: (b) Dynamic MRI Tracking according to updated important timestamps, and (c) MRI-Centric Scoring during decision phases, where tokens predicted to be critical for future steps are retained. (d) The MRI-Centric Score fundamentally predicts future token importance by analyzing historical patterns of importance variation (i.e., MRI and the time elapsed since their latest timestamp).

LazyEviction adopts a window size W as the interval for KV eviction decisions. As illustrated in Fig. 4(a), during decision-making, LazyEviction always retains the most recent W KVs to preserve local coherence. When the number of KVs exceeds the predefined budget $B (B \gg W)$, LazyEviction executes KV eviction. The overall procedure of our LazyEviction is summarized in Appendix. B.

To select $B - W$ KV pairs from past caches to retain, LazyEviction tracks importance variations (i.e., MRI) of each token at every decoding step (**Recurrence Interval Tracking**). For eviction decisions, importance scores, which are defined to indicate tokens' potential future importance, are calculated based on MRI. Consequently, LazyEviction selectively retains only the tokens in the KV cache most likely to be critical in the future (**MRI-Centric Eviction**).

Recurrence Interval Tracking. To identify importance variations for recurring tokens, we employ timestamps to track each token's time of being important. Consistent with RaaS (Hu et al., 2025), whenever a token receives an attention score exceeding a threshold α, its latest timestamp is updated to the value of current decoding step t. Thus, during the whole decoding phase, LazyEviction dynamically maintains a timestamp vector $\mathbf{TS}_t$ for all retained tokens, which records only the most recent activation time of tokens (illustrated in the leftmost figure in Fig. 4(b)).

However, only relying on $\mathbf{TS}_t$ fails to capture the temporal patterns of Importance Recurrence. To address this, we introduce MRI to record the longest interval between two consecutive activations in history. For newly generated tokens, MRI is initialized to 0. The MRI value is updated as:

$$\mathbf{MRI}_t = \max\{\mathbf{MRI}_{t-1}, \mathbf{TS}_t - \mathbf{TS}_{t-1}\}. \quad (1)$$

MRI-Centric Eviction. When the size of cached KV pairs $|\mathcal{S}_t|$ exceeds B, KV eviction are triggered at periodic intervals $t = kW$ ($k \in \mathbf{N}^+$). Leveraging the tracked historical importance dynamics via $\mathbf{MRI}_{t-1}$ and $\mathbf{TS}_t$. We aim to predict each token's future importance using an MRI-Centric Importance Score. As illustrated in Fig 4(d), the importance score should be designed under the following two intuitive heuristics:

1. **H1-score:** Given a token i, if the time elapsed since its last activation $(t - TS_t[i])$ is greater than its $MRI_t[i]$, the possibility of token i to be important in the future will decrease. The larger $\frac{t-TS_t[i]}{MRI_t[i]}$, the less likely token i is important. Thus, we define $\mathbf{H}_t^{(1)}[i] = 2\sigma(-\frac{t-TS_t[i]}{MRI_t[i]})$, where $\sigma(x) = 1/(1+\exp(-x))$ is sigmoid function.

2. **H2-score:** Tokens with smaller MRI values have a higher possibility to be important in the future, as frequently recurring tokens are deemed more critical. Thus, we define $\mathbf{H}_t^{(2)}[i] = 2\sigma(-MRI_t[i] - 1)$. Notably, if $MRI_t[i] = 0$, the token has never been activated since its generation (MRI is initialized as 0), resulting in $\mathbf{H}_t^{(2)}[i] = 0, \text{if} \mathrm{MRI_t[i]} = 0$.

H1-Score reflects the likelihood of the token regaining importance within the next observation window. It actually represents the token's "Survival Probability", as ΔT approaches MRI, it indicates that the probability of this token becoming important again decreases non-linearly. **H2-Score** prioritizes tokens with smaller MRI values, which represents the "Frequency Prior" of token recurrence. A shorter MRI implies that the token recurs more frequently, resulting in higher confidence that it is a critical token. We discuss the importance of these two score functions in Sec 5.4, and explain the rationale behind the formulation of each function in Appendix D. Thus, the MRI-Centric Importance Score can be formalized as:

$$\mathbf{I}_t[i] = \begin{cases} \mathbf{H}_t^{(1)}[i] + \mathbf{H}_t^{(2)}[i], & \text{if} MRI_t[i] \neq 0 \\ \mathbf{H}_t^{(1)}[i], & \text{if} MRI_t[i] = 0 \end{cases} \tag{2}$$

For each eviction decision, we first compute $\mathbf{I}_t$ with Eq. 2, then retain the most recent W and $\mathrm{Top(B-W)}$ KV pairs with the highest scores.

Notably, LazyEviction strategy introduces additional computational overhead, which may affect inference efficiency. We analyze this in Appendix E through both theoretical analysis and empirical evaluation. We point out that LazyEviction mitigates additional costs by making eviction decisions at fixed intervals rather than continuously. This W-steps eviction mechanism results in significantly lower overhead compared to existing KV compression methods. Moreover, since LazyEviction limits the KV budget during decoding steps, as the number of generated tokens increases, the computational efficiency does not grow linearly like it does with Full KV. When the generated tokens reach 16k, the overall inference efficiency of LazyEviction surpasses Full KV.

Disscussion about the choice of W and α. In LazyEviction, the observation window with size W represents the "safety buffer" for recurring tokens. Since critical tokens often go dormant for a specific number of steps before being reactivated, W is empirically set to cover the "dormant period" (MRI) of the majority of recurring tokens, preventing them from being evicted right before they are needed again. We employ the observation from Fig. 3(c), setting W as the MRI threshold corresponding to the 80% tokens. This threshold can be statistically determined through offline analysis by randomly selecting 1% samples from the test datasets. Our experiments in Appendix F.1 analyze how W affects reasoning performance.

The threshold α serves as a noise filter for the sparse attention mechanism, which distinguishes "active attention signals" from "background noise," ensuring that we only update timestamps for tokens that are genuinely being attended to. As the method for determining α has been thoroughly discussed in RaaS (Hu et al., 2025), we directly adopt their approach in LazyEviction. We analyze the effect of setting different α in Appendix F.2.

Table 1: Performance of our LazyEviction compared with baselines on mathematical reasoning dataset with DS-Llama-8B, DS-Qwen-7B, Qwen3-4B and QwQ-32B. The best results among all methods are in **bolded**.

Methods	DS-Llama	DS-Qwen	Qwen3	QwQ
GSM8K (Compression Ratio $r = 50\%$)				
FullKV	81.73	89.92	93.32	95.61
RaaS	78.01	85.37	90.22	84.31
H2O	77.16	87.04	88.47	89.23
TOVA	72.25	80.52	85.13	72.10
CAKE	78.29	64.43	76.08	67.42
R-KV	78.69	88.33	**91.65**	**93.63**
Ours	**80.06**	**88.40**	91.50	93.48
MATH-500 (Compression Ratio $r = 50\%$)				
FullKV	74.8	86.0	87.2	87.2
RaaS	71.2	82.4	84.0	80.4
H2O	67.2	80.4	79.6	80.4
TOVA	69.6	74.8	77.6	70.4
CAKE	53.8	59.2	62.8	54.6
R-KV	73.2	83.6	83.4	85.2
Ours	**75.2**	**85.4**	**85.8**	**85.4**
AIME (Compression Ratio $r = 30\%$)				
FullKV	30.0	46.7	60.0	73.3
RaaS	23.3	36.7	46.7	53.3
H2O	26.7	33.3	40.0	53.3
TOVA	23.3	36.7	40.0	36.7
R-KV	26.7	**43.3**	46.7	56.7
Ours	**30.0**	**43.3**	**53.3**	**66.7**

5 Experimental Results

In this section, we adopt variants of the DeepSeek-R1 distilled model (Guo et al., 2025; DeepSeek-AI et al., 2025): DeepSeek-R1-Distill-Llama-8B (DS-Llama-8B) and DeepSeek-R1-Distill-Qwen-7B (DS-Qwen-7B), and Qwen-series reasoning model: Qwen3-4B (Yang et al., 2025) and QwQ-32B (Team, 2024). The evaluations are carried on five reasoning benchmarks from three different domains, including mathematical reasoning (GSM8K (Cobbe et al., 2021), MATH-500 (Hendrycks et al., 2021), AIME (AIME)), Sci-

Table 2: Performance of our LazyEviction compared with baselines on GPQA Diamond and LiveCodeBench with DS-Llama-8B and DS-Qwen-7B. The best results among all methods are in **bolded**.

Methods	DS-Llama-8B	DS-Qwen-8B
GPQA Diamond ($r = 50\%$)		
FullKV	37.4	55.7
RaaS	29.9	45.4
H2O	26.8	42.1
TOVA	30.9	44.3
R-KV	28.2	41.2
LazyEviction	**36.9**	**54.6**
LiveCodeBench ($r = 40\%$)		
FullKV	58.62	55.17
RaaS	53.45	39.66
H2O	53.45	50.00
TOVA	51.72	29.31
R-KV	48.28	46.55
LazyEviction	**56.90**	**51.72**

ence QA (GPQA Diamond (Rein et al., 2024)) and programming (LiveCodeBench (Jain et al., 2024)).

We compare LazyEviction with FullKV and four representative KV cache compression methods, including TOVA (Oren et al., 2024), H2O (Zhang et al., 2023), CAKE (Qin et al., 2025), RaaS (Hu et al., 2025), and R-KV (Cai et al., 2025). We use the KV compression ratio r to denote the KV cache budget across different baselines, where r represents the proportion of KV cache usage relative to FullKV. More implementation details can be found in Appendix C.

5.1 Performance Comparison with Baselines

Table 1 and Table 2 present a comprehensive performance comparison of our LazyEviction and baselines. The results demonstrate that LazyEviction consistently outperforms other methods across all datasets, achieving performance comparable to the baseline FullKV with minimal deviation. In mathematical reasoning tasks, LazyEviction achieves performance close to FullKV with only 30%~50% KV budgets. Additionally, LazyEviction also achieves the best performance across GPQA Diamond and LiveCodeBench datasets, demonstrating its applicability among different domain tasks. It is worth noting that even though R-KV achieves performance close to LazyEviction in mathematical reasoning tasks, its performance significantly declines on other datasets. This is due to R-KV's excessive reliance on the assumption that there are many similar tokens present in the reasoning path. In other tasks, similar tokens are not as prevalent as in mathematical tasks.

Trade-off between Accuracy and KV Cache Size. We evaluate the performance of LazyEviction in optimizing the trade-off between KV cache budget and accuracy compared to baselines. As shown in Fig. 5, our method maintains higher accuracy across various budgets for both datasets and models. When the cache budget is large, other methods perform slightly worse than our approach. However, with a smaller cache budget, other methods experience substantial accuracy degradation. By implementing lagged KV eviction and leveraging MRI to track recurring tokens, our approach consistently achieves optimal performance, even under stringent budget constraints. Notably, on MATH-500 and AIME datasets, LazyEviction even outperformed FullKV's performance while reducing the KV cache budget by 50%.

5.2 Memory Efficiency of LazyEviction

We evaluate different methods on DS-Qwen-7B model, analyzing their KV cache memory usage variation with output length. As shown in Fig. 6, the memory consumption of FullKV increases linearly with the number of tokens. In contrast, the memory usage of TOVA, H2O, and RaaS methods remains constant once the token count exceeds the budget. LazyEviction, with the observation window mechanism, exhibits minor fluctuations in KV cache memory usage after surpassing the budget, but Fig. 6 indicates that its memory growth is minimal compared to the other three algorithms.

5.3 Generalizability to General Long-form Generation Scenarios

While the main text primarily focuses on reasoning-intensive tasks (e.g., Math, Coding) where long-range dependencies and backtracking are most prominent, LazyEviction is fundamentally designed as a general framework for *Long-form Generation*. In this section, we extend our evaluation to mainstream QA and long-document summarization tasks to demonstrate the ubiquity of the Token Importance Recurrence (TIR) phenomenon and the robustness of our approach.

Experimental Setup for General Tasks To evaluate the performance on general tasks, the experiments are carried on the non-reasoning model adopt Llama-3.1-8B-Instruct. We target scenarios where the GPU memory overhead from the

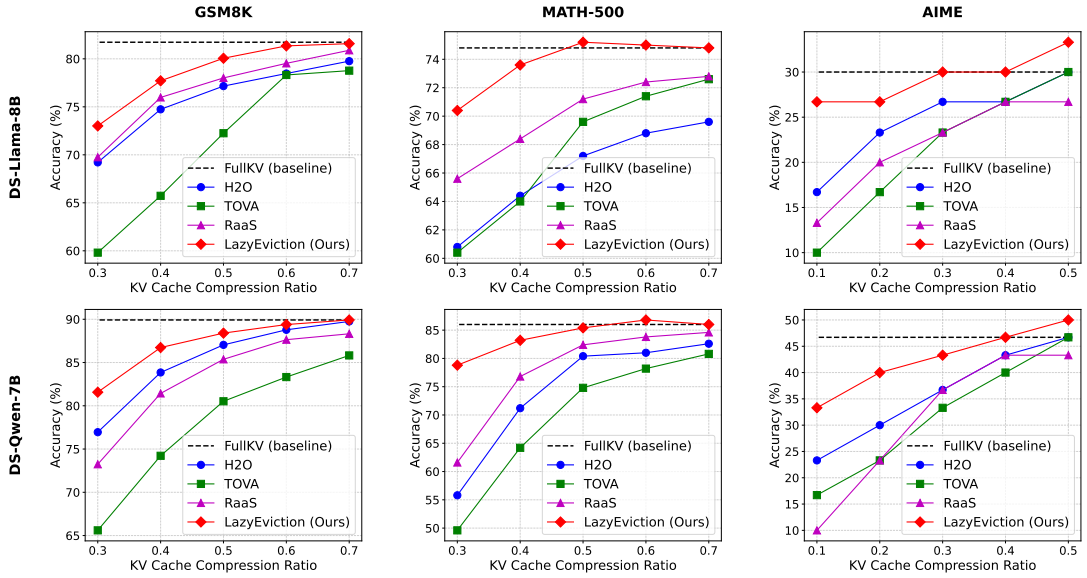


Figure 5: Trade-off between accuracy and KV cache size among different datasets and models.

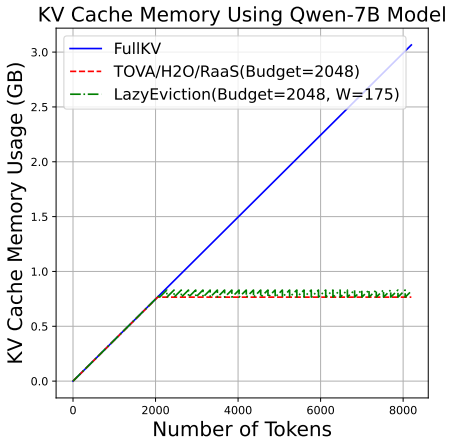


Figure 6: KV cache memory usage of different methods with varying output length (0-8k tokens).

KV cache becomes a bottleneck, specifically tasks requiring long-sequence generation. Traditional short-answer QA tasks (generating less than 5 tokens) are excluded as they do not incur significant memory pressure. We use the following representative benchmarks:

- **Mainstream QA (PIQA (Bisk et al., 2020)):** We utilize specific prompting (e.g., adding `"Let's think step by step"`) to guide the model into generating explanatory answers (CoT-style), aligning the task with real-world long-form generation.

- **Long-form Generation (SQuALITY (Wang et al., 2022)):** A long-document summarization dataset that naturally involves extensive context and long-form output generation.

We analyze the distribution of Maximum Recurrence Interval (MRI) on these datasets with Llama-3-8B-Instruct. We observe that the TIR phenomenon remains prevalent in general long-form generation, although the magnitude differs from reasoning tasks.

Compared to Math/Code tasks (e.g., GSM8K), the MRI in non-reasoning tasks is relatively smaller. For example, only 20% tokens' MRI ≥ 5 observed on SQuALITY datasets, while more than 60% tokens have larger MRI on math reasoning datasets like GSM8K. This because summarization and general QA typically require less "reflection" or "long-range backtracking". However, most tokens' (>90%) importance still exhibits "non-monotonic" fluctuations (MRI > 1).

Performance Comparison We compare LazyEviction against TOVA and H2O on PIQA and SQuALITY. We specifically chose compression ratio $r = 30\%$ for PIQA and $r = 20\%$ for SQuALITY because these are the critical thresholds where baseline methods typically fail.

The results demonstrate that even under these aggressive compression settings (where baselines struggle), LazyEviction consistently achieves superior Accuracy and ROUGE scores compared to baselines. This indicates that traditional greedy algorithms (e.g., TOVA, H2O) tend to evict tokens during these temporary periods of low attention, leading to information loss when the tokens are

Table 3: Performance of our LazyEviction compared with baselines on PIQA and SQuALITY with Llama-3-8B-Instruct. The best results among all methods are in **bolded**.

Methods	*PIQA*($r = 30\%$)	*SQuALITY*($r = 20\%$)
	Acc. (↑)	ROUGE (↑)
FullKV	73.00	14.93
H2O	64.72 (-8.28)	14.69 (-0.24)
TOVA	58.48 (-14.52)	14.03 (-0.90)
Ours	**71.50 (-1.50)**	**15.05 (+0.12)**

Table 4: Baseline algorithms with observation window mechanism, evaluating on the GSM8K dataset using DS-Llama-8B with $r = 50\%$ and $W = 25$.

Model	DS-Distill-Llama-8B
LazyEviction	80.06
H2O	77.16
+ window	78.88 (+1.72)
TOVA	72.25
+ window	76.09 (+3.84)
RaaS	78.01
+ window	78.92 (+0.91)

needed later. In contrast, LazyEviction's Observation Window mechanism tolerates such short-term low attention, ensuring the retention of latent recurring tokens and significant robustness improvements over baselines.

In conlusion, our method does not solely rely on the ultra-long MRI characteristic found in reasoning tasks. As long as fluctuations in token importance exist, the "lagged observation + dynamic prediction" mechanism provides a more robust solution than greedy eviction. This confirms that LazyEviction achieves SOTA performance in general long-form generation tasks, demonstrating that LazyEviction is not merely a reasoning-specific design, but a robust, general-purpose KV eviction strategy capable of handling attention noise better than existing greedy approaches.

5.4 Ablation Study

Lagged Eviction with an Observation Window Since our method retains a slightly increased number of KV pairs within the observation window, which may lead to the perceived performance gains being predominantly attributed to these additional tokens. To eliminate the impact of the additional tokens within observation window, we explore the other algorithms with the observation window mechanism on performance. As shown in Table 4, when H2O, TOVA, and RaaS algorithms perform KV eviction every W decoding steps, their performance improves. This improvement arises from the linear increase in retained KV pairs between consecutive decision decoding steps. However, their performance remains inferior to that of LazyEviction. LazyEviction employs MRI to track recurring tokens, effectively capturing their potential future importance, which leads to superior performance.

Table 5: Performance with different importance scores.

Model	DS-Llama-8B	DS-Qwen-7B
LazyEviction	80.06	88.40
w/o the **H1-Score**	76.11 (-3.95)	82.78 (-5.62)
w/o the **H2-Score**	79.67 (-0.39)	87.21 (-1.19)

The Discussion of the Importance Score We further investigate the influence of MRI-Centric Importance Score, as shown in Table 5. If the importance score $\mathbf{I}_t$ calculation excludes the **H1-Score**, accuracy significantly decreases, underscoring the effectiveness of the recurrence interval tracking mechanism. Similarly, omitting the **H2-Score** leads to a slight accuracy drop, indicating that tokens with smaller MRI values are indeed more likely to become important in the future. Collectively, these findings demonstrate the effectiveness of LazyEviction's MRI-Centric Eviction Policy.

6 Conclusion

In this work, we address the critical bottleneck of KV cache memory overhead in long reasoning tasks. Our analysis reveals the Token Importance Recurrence phenomenon, where certain significant tokens receive renewed attention after multiple decoding steps. Based on this insight, we propose LazyEviction, a novel lagged eviction framework that introduces temporal observation windows, recurrence interval tracking, and an MRI-centric eviction policy to preserve recurring tokens while dynamically reducing memory usage. Experimental results show that LazyEviction achieves performance close to or outperforms FullKV with only 30%~50% KV budgets, highlighting the importance of preserving recurring tokens on maintaining knowledge continuity in reasoning tasks. By shifting from greedy eviction to predictive retention, our work provides a new paradigm for efficient KV cache compression in long reasoning scenarios.

Limitations

Dynamically Adjusting in Observation Window Size. We discuss in Appendix F.1 that the suitable observation window size is crucial for identifying recurring tokens. In Sec. 3, we find that the MRI distribution of recurring tokens varies across different models and tasks. Therefore, we propose to offline select 1% of samples to pre-statistically analyze the MRI distribution to determine the observation window size. However, this approach is inefficient and sub-optimal in practice. Thus, dynamically and adaptively adjusting the observation window based on task characteristics will be an important direction for improvement in LazyEviction in the future.

While our current experiments utilize offline MRI statistics to determine the observation window size W, we emphasize that the selection of W is a replaceable component within the LazyEviction paradigm, rather than a theoretical constraint. The core contribution of this work is the identification of the TIR phenomenon and the lagged eviction framework. The offline statistic is merely one implementation to estimate the recurrence interval.

For real-world deployments where offline test data is unavailable or data leakage is a concern, our framework supports flexible adaptation strategies: (1) **Calibration Set:** In practice, W can be determined using a small validation set or synthetic prompts without accessing the test data, as MRI reflects general model-task dynamics. (2) **Online Warm-up:** We propose an online warm-up strategy where the system initially employs a conservative window for the first few requests to profile the MRI distribution in the background. As demonstrated in Appendix F.3, reliable statistics can be gathered within minutes, allowing the system to update to an optimal W dynamically. This completely eliminates the dependency on offline statistics and ensures zero data leakage.

Evaluation Beyond Reasoning Tasks. Although we have conducted experiments on tasks from different domains, our evaluation is still limited to reasoning tasks. This is because the core finding of this paper, the Token Importance Recurrence, is particularly significant in reasoning scenarios. We attempt to investigate TIR in non-reasoning scenarios. For example, we use Llama-3.1-8B-Instruct[1] to analyze the MRI distribution on the language modeling dataset C4[2]. The results show that although TIR also exists, recurring tokens have a relatively smaller MRI (<10). In this case, the performance of LazyEviction is not significantly different from that of Cumulative Attention-based Eviction methods (e.g., H2O, RaaS). Therefore, we clarify that the observations and methods presented in this paper are more applicable to reasoning scenarios.

Evaluation on Larger Scale Models. Our experiments cover reasoning models ranging from 4B to 32B in size. However, due to limitations in the experimental environment, we have not explored larger scale models, such as Qwen3-Max and DeepSeek-R1. Inference on models at the 100B level is extremely time- and resource-consuming. For example, evaluating 500 samples MATH-500 with DeepSeek-R1 requires days on 8 A100 GPUs. Nonetheless, we believe that our core finding, the Token Importance Recurrence phenomenon, and the importance of preserving recurring tokens exist across all reasoning models, and we will extend our experiments to more models in the future.

[1]https:https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct

[2]https://huggingface.co/datasets/allenai/c4

Acknowledgments

This research was supported by fundings from the Hong Kong RGC General Research Fund (152228/23E, 162161/24E, 162116/25E, 162180/25E), National Natural Science Foundation of China (NSFC) Key Program (No.62532005), Collaborative Research Fund (No. C1042-23GF, No. C5097-25G), NSFC/RGC Collaborative Research Scheme (Grant No. 62461160332 & CRS_HKUST602/24), Research Impact Fund (No. R5011-23F), Areas of Excellence Scheme (AoE/E-601/22-R), and the InnoHK (HKGAI).

References

Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant J Nair, Ilya Soloveychik, and Purushotham Kamath. 2024. Keyformer: Kv cache reduction through key tokens selection for efficient generative inference. *Proceedings of Machine Learning and Systems*, 6:114–127.

AIME. Aime_1983_2024.

Simon A Aytes, Jinheon Baek, and Sung Ju Hwang. 2025. Sketch-of-thought: Efficient llm reasoning with adaptive cognitive-inspired sketching. *arXiv preprint arXiv:2503.05179*.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, and Yejin Choi. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

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. 2025. R-kv: Redundancy-aware kv cache compression for training-free reasoning models acceleration. *arXiv preprint arXiv:2505.24133*.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. 2024. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Yilong Chen, Guoxia Wang, Junyuan Shang, Shiyao Cui, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, Yu Sun, Dianhai Yu, and Hua Wu. 2024. Nacl: A general and effective kv cache eviction framework for llms at inference time. *arXiv preprint arXiv:2408.03675*.

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. 2021. Training verifiers to solve math word problems. *URL https://arxiv. org/abs/2110.14168*, 9.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 181 others. 2025. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. 2023. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Advances in Neural Information Processing Systems*, 36:70757–70798.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. 2024. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *arXiv preprint arXiv:2407.11550*.

Yu Fu, Zefan Cai, Abedelkadir Asi, Wayne Xiong, Yue Dong, and Wen Xiao. 2024. Not all heads matter: A head-level kv cache compression method with integrated retrieval and reasoning. *arXiv preprint arXiv:2410.19258*.

Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D Goodman. 2025. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars. *arXiv preprint arXiv:2503.01307*.

Ravi Ghadia, Avinash Kumar, Gaurav Jain, Prashant Nair, and Poulami Das. 2025. Dialogue without limits: Constant-sized kv caches for extended responses in llms. *arXiv preprint arXiv:2503.00979*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2024. Token-budget-aware llm reasoning. *arXiv preprint arXiv:2412.18547*.

Jitai Hao, Yuke Zhu, Tian Wang, Jun Yu, Xin Xin, Bo Zheng, Zhaochun Ren, and Sheng Guo. 2025. Omnikv: Dynamic context selection for efficient long-context llms. In *The Thirteenth International Conference on Learning Representations*.

Ziwei He, Jian Yuan, Haoli Bai, Jingwen Leng, and Bo Jiang. 2025. Treekv: Smooth key-value cache compression with tree structures. *arXiv preprint arXiv:2501.04987*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Yuhan Dong, and Yu Wang. 2023. Flashdecoding++: Faster large language model inference on gpus. *arXiv preprint arXiv:2311.01282*.

Junhao Hu, Wenrui Huang, Weidong Wang, Zhenwen Li, Tiancheng Hu, Zhixia Liu, Xusheng Chen, Tao Xie, and Yizhou Shan. 2025. Efficient long-decoding inference with reasoning-aware attention sparsity. *arXiv preprint arXiv:2502.11147*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in*

neural information processing systems, 35:22199–22213.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970.

Yongjiang Liu, Haoxi Li, Xiaosong Ma, Jie Zhang, and Song Guo. 2025. Think how to think: Mitigating overthinking with autonomous difficulty cognition in large reasoning models. *Preprint*, arXiv:2507.02663.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364.

OpenAI, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, and 243 others. 2024. Openai o1 system card. *Preprint*, arXiv:2412.16720.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. 2024. Transformers are multi-state rnns. *arXiv preprint arXiv:2401.06104*.

Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. 2025. Cake: Cascading and adaptive kv cache eviction with layer preferences. *arXiv preprint arXiv:2503.12491*.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. 2019. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

Jiwon Song, Dongwon Jo, Yulhwa Kim, and Jae-Joon Kim. 2025. Reasoning path compression: Compressing generation trajectories for efficient llm reasoning. *arXiv preprint arXiv:2505.13866*.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*.

Qwen Team. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2:3.

Alex Wang, Richard Yuanzhe Pang, Angelica Chen, Jason Phang, and Samuel Bowman. 2022. Squality: Building a long-document summarization dataset the hard way. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1139–1156.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

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

Heming Xia, Yongqi Li, Chak Tou Leong, Wenjie Wang, and Wenjie Li. 2025. Tokenskip: Controllable chain-of-thought compression in llms. *arXiv preprint arXiv:2502.12067*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

Yuchen Yan, Yongliang Shen, Yang Liu, Jin Jiang, Mengdi Zhang, Jian Shao, and Yueting Zhuang. 2025. Inftythink: Breaking the length limits of long-context reasoning in large language models. *arXiv preprint arXiv:2503.06692*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Jintian Zhang, Yuqi Zhu, Mengshu Sun, Yujie Luo, Shuofei Qiao, Lun Du, Da Zheng, Huajun Chen, and Ningyu Zhang. 2025. Lightthinker: Thinking step-by-step compression. *arXiv preprint arXiv:2502.15589*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and Beidi

Chen. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

A Mathematical Formulation of LazyEviction

Suppose the sequence length is N_t at decoding step t. We denote $\mathbf{Q}_i$, $\mathbf{K}_i$ and $\mathbf{V}_i$ as the query, key and value vectors of the i-th token. The attention output at the decoding step t can be formulated as:

$$\mathbf{A}_t(\mathbf{Q}_{N_t}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}_{N_t}\mathbf{K}^T}{\sqrt{d_h}}\right)\mathbf{V}, \tag{3}$$

where $\mathbf{K} = [\mathbf{K}_1, ..., \mathbf{K}_{N_t}] \in \mathbb{R}^{N_t \times d_h}$, $\mathbf{V} = [\mathbf{V}_1, ..., \mathbf{V}_{N_t}] \in \mathbb{R}^{N_t \times d_h}$ and d_h is the head dimension. We define $\mathcal{S}_t$ as the set of the KV pairs $(\mathbf{K}_i, \mathbf{V}_i)$ for all the previous tokens $i < N_t$ cached in GPU memory at decoding step t. Since the size of $\mathbf{K}$ and $\mathbf{V}$ grows linealy with the decoding step t, current methods try to evict the unimportant KV pairs from $\mathcal{S}_t$ at each step t. Unlike the previous KV compression methods, we make KV eviction decisions at decoding step $t = kW$ ($k \in \mathbf{N}^+$). Denote $\mathcal{S}_t^*$ as the optimal KV pairs selected from $\mathcal{S}_t$ under the cache budget B. Thus, the KV eviction problem can be formulate as:

$$\mathcal{S}_t^* = \underset{|\mathcal{S}_t'|=B, \mathcal{S}_t' \subseteq \mathcal{S}_t}{\arg\min} \left\| \mathbf{A}_t(\mathbf{Q}_{N_t}, \mathbf{K}, \mathbf{V}) - \mathbf{A}_t'(\mathbf{Q}_{N_t}, \mathbf{K}', \mathbf{V}', \mathcal{S}_t') \right\|_2 \tag{4}$$

where $\mathbf{K}'$ and $\mathbf{V}'$ are selected KV pairs given by $\mathcal{S}_t'$, $\mathbf{A}_t'$ is the attention output computed by Eq. 3 after KV eviction. To minimize the effect of the KV eviction process, our goal is to find the optimal $\mathcal{S}_t^*$ to minimize the change in attention output.

It is intractable to directly solve Eq. 4 because of its combinatorial nature. We design a heuristic method to efficiently find the suboptimal solution for $\mathcal{S}_t^*$. First, as emphasized in previous works (Xiao et al., 2023; Zhang et al., 2023; Ghadia et al., 2025), retaining recent tokens is critical to ensuring coherence in the generated context. And to observe the importance variations of the newly generated tokens, we consistently preserve the latest W KV pairs. Second, among the remaining past KVs, we retain $B - W$ KV pairs by prioritizing tokens most likely to exhibit importance recurrence in future steps. This selection is guided by an auxiliary score vector $\mathbf{I}_t$, which quantifies the potential future importance of tokens. Formally, our eviction policy is defined as:

$$\mathcal{S}_t' = \text{Top}_{[\text{B}-\text{W}]}(\mathbf{I}_\text{t}) \cup \mathcal{W}_\text{t} \tag{5}$$

where $\text{Top}_{[\text{k}]}(\cdot)$ represents the selection of top-k entries (KV pairs) from $\mathcal{S}_t$ and $\mathcal{W}_t$ is the set of the most W recent KV pairs in $\mathcal{S}_t$.

Algorithm 1 LazyEviction

Require: $\mathbf{A}_t$, $\mathbf{K}$, $\mathbf{V}$, W, B, α, t.

```
for All token i do
    if A_t[i] >= α then
        Update TS_t[i] = t.
    end if
end for
Update MRI_t according to Eq. 1.
Calculate I_t according to Eq. 2.
if t == kW and B >= |K| then
    Update KV pairs according to Eq. 5.
end if
return S'_t.
```

B Overall Procedure of LazyEviction

We provide the pseudo-code of LazyEviction in Algorithm. 1, which contains **Recurrence Interval Tracking** at each step t and **MRI-Centric Eviction** at the decoding step $t = kW$ ($k \in \mathbf{N}^+$). Thus, Lazyeviction tasks the current decoding step t, observation window size W, cache budget B, α, and $\mathbf{A}_t$, $\mathbf{K}$, $\mathbf{V}$ as input.

C Experiment Setup

Models and Datasets By fine-tuning with CoT data generated by DeepSeek-R1, even smaller distilled models can acquire long-reasoning capabilities similar to DeepSeek-R1 (Guo et al., 2025). Thus, we first adopt DeepSeek-R1-Distill-Llama-8B and DeepSeek-R1-Distill-Qwen-7B as the base models for evaluating KV cache compression. In addition, to validate the effectiveness of LazyEviction across a broader range of model types, we also utilize additional Qwen-series reasoning models with different sizes: Qwen3-4B (Yang et al., 2025) and QwQ-32B (Team, 2024).

The evaluations are mainly carried on three widely used mathematical reasoning benchmarks: GSM8K(Cobbe et al., 2021), MATH-500(Hendrycks et al., 2021) and AIME (AIME).

GSM8K is a high-quality, linguistically diverse dataset of elementary-level math problems requiring multi-step reasoning and basic arithmetic operations to solve. MATH-500 contains 500 challenging problems drawn from high school math competitions, spanning domains such as algebra, geometry, and number theory. AIME is a math problem dataset collected from the American Invitational Mathematics Examination, designed to challenge the most exceptional high school mathematics students in the United States.

Besides mathematical reasoning benchmarks, we also adopted reasoning benchmarks from other domains to validate the generalization of LazyEviction across different domain tasks. Specifically, GPQA Diamond (Rein et al., 2024) is a multiple-choice science QA dataset covering the fields of biology, physics and chemistry, while LiveCodeBench (Jain et al., 2024) is an advanced evaluation benchmark (1116 samples) used to rigorously assess the code processing capabilities of LLMs. Since it may cost days to evaluate the entire benchmark, we randomly sample 5% data (58 samples) from the dataset for evaluation.

Baselines To validate the performance of LazyEviction, we compare it against Full Cache and three representative KV cache compression methods, including TOVA (Oren et al., 2024), which retains tokens with the highest attention scores at each decoding step; H2O (Zhang et al., 2023), which preserves heavy hitter tokens (identified by cumulative attention scores) and recent tokens during decoding; RaaS (Hu et al., 2025), which selects tokens with the newest timestamps at every decoding step; and R-KV (Cai et al., 2025), which utilize the similarity among tokens to identify and evict redundant tokens in long reasoning tasks.

Implementation Details We implement LazyEviction in HuggingFace transformers library (Wolf et al., 2020) to integrate it into the existing attention mechanism. We set the observation window size equal to the value of 80% MRI threshold based on the preliminary observations (Fig. 3(c)), and the number of recent tokens in H2O is equal to LazyEviction's window size. Unless stated otherwise, we set $\alpha = 0.0005$ for DS-Llama-8B model and $\alpha = 0.0001$ for DS-Qwen-7B, Qwen-4B and QwQ-32B models. Experiments are conducted on NVIDIA V100 (32GB) GPUs. We use Ubuntu 20.04 with Linux kernel 6.8.0 and CUDA 12.8. The maximum number of generated tokens are set as 4096 for GSM8K dataset, 8192 for MATH-500 and GPQA datasets, 16384 for AIME and LiveCodeBench datasets, respectively.

Table 6: Measurement of average decoding latency and throughput for LazyEviction compared with baselines.

	H1-score	H2-score
	GSM8K	
Initial	Sigmoid: 88.40	Sigmoid: 88.40
Change to	Exp: 88.25 (-0.15) Tanh: 88.25 (-0.15) Log: 88.40 (-) Inverse: 88.40 (-)	Exp: 88.25 (-0.15) Tanh: 88.33 (-0.07) Log: 88.33 (-0.07) Inverse: 88.40 (-)
	MATH-500	
Initial	Sigmoid: 85.4	Sigmoid: 85.4
Change to	Exp: 85.2 (-0.2) Tanh: 85.2 ((-0.2) Log: 85.0 (-0.4) Inverse: 85.0 (-0.4)	Exp: 85.4 (-) Tanh: 84.2 (-0.8) Log: 84.8 (-0.6) Inverse: 85.2 (-0.2)

D Discussion about formulations of the importance score.

As illustrated in Sec 4, both H1-score and H2-score are formulated by sigmoid functions, which performs as "Soft Gating" mechanism and best fits the non-linear decay law of token importance. The choice of the formulations have been thoroughly considered, where the score function must possess the following two properties: (1) The score function needs to have a monotonically decreasing property; (2) The score values must fall within the range of [0, 1]. For functions that meet these criteria, we have tried exponential, logarithmic, inverse, sigmoid, and tanh functions. In preliminary experiments, we found that the sigmoid function yielded the most stable results and was more concise in form, leading us to define the importance scores.

We evaluated different combinations of functions on the GSM8K and MATH-500. As shown in Table 6, each function forms perform similarly. Among them, the sigmoid functions achieves a better and stable performance. Therefore, we adopt sigmoid function for H1- and H2-score.

E Computational Cost of LazyEviction

It is important to notice that LazyEviction introduces additional computational overhead, which may affect inference efficiency. Therefore, it is necessary to analyze the additional computational introduced by LazyEviction. We first clarify that

Table 7: Computational complexity for each eviction algorithms within one window.

Method	Computational Complexity
H2O	$O[W(B + BlogB)]$
TOVA	$O(WBlogB)$
RaaS	$O[W(B + BlogB)]$
LazyEviction	$O(WB + BlogB)$

all KV eviction strategies inherently incur computational overhead, as they require decision-making at each decoding step. However, LazyEviction mitigates this cost by performing eviction decisions at fixed intervals, rather than continuously. This W-steps eviction mechanism leads to significantly lower overhead compared to baseline methods, which we will demonstrate through both theoretical analysis and empirical evaluation.

E.1 Theoretical Analysis of Computational Complexity

Unlike existing works that perform KV eviction decisions at every decoding step, LazyEviction's Observation Window-based Lagged Eviction adopts a window W as the interval for KV eviction decisions. Therefore, we analyze computational complexity over W decoding steps. All KV cache eviction algorithms can be divided into two steps: importance score calculation and importance ranking. We only consider the additional computational overhead, where the computation of attention process is not involved. Given a KV cache budget B, the comparison between LazyEviction and baseline methods is shown in Table 7.

Here, $BlogB$ is the computational complexity of a single importance ranking. Compared to LazyEviction, baseline methods require W importance rankings within W steps. For importance score calculation, LazyEviction, like other baseline methods, requires a finite number of computations for B KV pairs (Eq 1 MRI updating and Eq 2 Importance score calculating). Since $W \gg 1$, the computational overhead of LazyEviction is minimal compared to baseline methods.

E.2 Experimental Computational Overhead Measurement

Single-Step Decoding Latency and Trade-offs. We evaluate the single-step decoding latency to analyze the trade-off between the computational overhead introduced by LazyEviction and the efficiency gains from KV cache compression. The

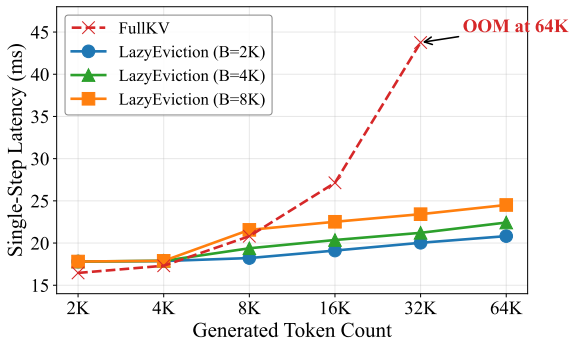


Figure 7: Measurement of single-step decoding latency (ms) under different KV cache budgets. LazyEviction maintains constant latency after the cache is full, preventing the unbounded growth seen in FullKV.

experiment involves generating a sequence of up to 64k tokens.

As shown in Fig 7, FullKV exhibits a linear growth in latency as the sequence length increases, eventually leading to Out-Of-Memory (OOM) errors at 64k tokens. In contrast, LazyEviction demonstrates only a slight increase in latency once the KV cache reaches the predefined budget B.

When the generated sequence length is smaller than the cache budget ($t < B$), LazyEviction incurs a marginal computational overhead compared to FullKV (e.g., generated token count=2K and 4K). This is because LazyEviction performs MRI tracking and timestamp updates at every step to learn attention patterns, even though no eviction is triggered yet. We consider this a necessary cost to ensure accurate identification of recurring tokens.

Once the sequence length exceeds the budget ($t > B$), the benefits of LazyEviction become evident. While FullKV's latency continues to grow unboundedly, LazyEviction maintains a nearly constant latency. This confirms that LazyEviction is particularly advantageous for memory-constrained scenarios where preventing OOM and curbing latency growth is prioritized over short-sequence speed.

Average Decoding Latency and Throughput We illustrate that LazyEviction introduces the least additional computational overhead by comparing average decoding latency and throughput. In practical LLM serving, the most effective way to increase throughput is to maximize the batch size. Since FullKV fills GPU memory rapidly with long reasoning chains (up to 16k tokens), it forces small batch sizes, creating a memory bottleneck. To demonstrate the practical efficiency of LazyEviction, we conducted a experiment measuring Maxi-

Table 8: Measurement of average decoding latency and throughput for LazyEviction. (Max Generation Length = 16K tokens)

Budget	Batch size	Throughput (token/s) ↑	Avg. Latency (ms/token) ↓
FullKV			
-	1	47.61	21.00
-	5 (max)	90.25	11.08
TOVA			
8192	1	43.94	22.76
8192	9 (max)	199.47	5.01
4915	16 (max)	347.28	2.88
LazyEviction			
8192	1	48.64	20.56
8192	9 (max)	213.54	4.68
4915	16 (max)	386.25	2.59

mum Throughput under memory-constrained settings, i.e., Single NVIDIA V100 (32GB).

We increased the batch size for each method until reaching the GPU memory limit. As shown in Table 8, when the batch size increases, FullKV is firstly constrained by the memory bottleneck. However, due to the lower KV budgets, LazyEviction can support a larger batch size, which achieves **2.37x** to **4.28x** the throughput of FullKV. Even though LazyEviction has a small computational overhead, this is vastly outweighed by the parallelization gains.

To measure the additional computational overhead of LazyEviction, we compare it with TOVA under different setting of output lengths and batch sizes. (Since TOVA does not introduce any additional computations aside from importance scoring, it has the smallest computational overhead among methods.) As shown in Table 8, LazyEviction outperforms TOVA at different output lengths and batch sizes, indicating that our LazyEviction introduces less additional computational overhead.

F Discussion about W and α

F.1 The Selection of Window Size W.

In LazyEviction, the observation window with size W serves to detect recurring tokens. However, since the most recent W-th tokens must be preserved, the setting of W inherently introduces a trade-off between local and global information retention. An excessively large W risks losing critical past tokens, while an overly small W may fail to detect numerous recurring tokens.

We investigate the impact of the window size W on DS-Llama-8B performance across two datasets in Table 9. As W increases, the model's accuracy generally improves. This is because a larger window size allows the LazyEviction mechanism's MRI metric to detect more recurring tokens, resulting in better capturing a token's potential future significance, thereby enhancing performance. However, when the window size becomes too large, LazyEviction risks discarding critical past tokens, leading to a slight performance decline. Thus, the choice of W presents a trade-off between retaining local and global information.

Table 9: Performance of different setting of the window size W on DS-Llama-8B.

Dataset	**GSM8K** ($r = 50\%$)				
	W=4	W=8	W=16	**W=25**	W=32
Acc.	75.14	76.58	76.73	**80.06**	79.63
Dataset	**MATH-500** ($r = 50\%$)				
	W=8	W=16	W=32	**W=52**	W=64
Acc.	71.8	72.2	72.8	**75.2**	74.8

Table 10: Performance of different setting of α.

Model	**DS-Llama-8B** ($r = 50\%$)			
	FullKV	α=0.0001	α=0.0005	α=0.001
Acc.	81.73	78.97	**80.06**	79.68
Model	**DS-Qwen-7B** ($r = 50\%$)			
	FullKV	α=0.00001	α=0.0001	α=0.001
Acc.	89.92	88.06	**88.40**	88.32

F.2 The Selection of α

We further explore the impact of the threshold α on the GSM8K dataset using two models. Since the selection of α influences the temporal distribution of token timestamps, both extremely large and small α hinder effective identification of tokens' importance variation. The value of α influences the MRI of each token, with $\alpha = 0.0005$ producing optimal results for DS-Llama-8B and $\alpha = 0.0001$ yielding the best performance for DS-Qwen-7B, as detailed in Table 10. First, when α is small, tokens are more readily flagged as important, but the MRI of these tokens is reduced, which does not accurately reflect the periodicity of their significance. Second, when α is large, the steps where tokens gain importance are not captured, preventing the detection of their recurring patterns.

Dataset	Process	Samples	Runtime
GSM8K	MRI Analysis	20	∼10 mins
(Avg. Len ∼1k)	Evaluation	1319	∼7 hours
MATH500	MRI Analysis	5	∼15 mins
(Avg. Len ∼2k)	Evaluation	500	∼20 hours
AIME	MRI Analysis	2	∼40 mins
(Avg. Len ∼10k)	Evaluation	30	∼10 hours

Table 11: Comparison of computational cost between MRI Analysis (for determining W) and full Evaluation. The overhead for determining W is minimal.

F.3 Efficiency of Determining W

While determining the optimal observation window size W requires analyzing MRI statistics, we demonstrate that this process incurs minimal computational overhead and is highly practical. The MRI distribution is jointly determined by the model's inherent characteristics and the task type (i.e., token-level attention dynamics). Due to the vast number of tokens generated in long-context tasks, these statistics converge extremely quickly.

As shown in Table 11, relying on a negligible number of samples (ranging from 2 to 20) or merely minutes of inference time is sufficient to obtain reliable MRI statistics for determining W. Compared to the runtime required for full dataset evaluation, this pre-statistical overhead is negligible, confirming the practicality of our offline analysis approach.

G Comparison between LazyEviction and RaaS

Previous work RaaS (Hu et al., 2025) shares a similar motivation with our work, as both focus on the dynamic changes in token importance during reasoning tasks. In summary, RaaS can be viewed as a special case of the LazyEviction strategy within "short-input, long-reasoning" scenarios (assuming Prefill is always important and Decode gradually fades). In contrast, LazyEviction provides a more robust mathematical framework (MRI) capable of adapting to a wider range of length distributions and attention patterns. We summarize the core differences below and include a detailed comparative discussion between LazyEviction and RaaS:

1. Difference in Definition of "Recurrence"

- **RaaS (Phoenix & Milestone):** RaaS categorizes tokens into two distinct types:
  - *Phoenix Tokens:* Refer specifically to tokens in the Prefill stage (e.g., problem conditions), which reactivate after a period of silence.
  - *Milestone Tokens:* Refer specifically to tokens in the Decode stage. RaaS assumes these follow a "Waterfall" pattern, where importance decays monotonically—once it fades, it never rises again.
- **LazyEviction (TIR):** Our proposed Token Importance Recurrence (TIR) is a unified phenomenon. Our research reveals that recurrence features (i.e., MRI > 0) are exhibited not only by Prefill tokens but also by tokens generated during decoding (intermediate reasoning steps), contrary to RaaS's assumption of monotonic decay. LazyEviction captures recurrence patterns across all positions without distinguishing the token source.

2. Difference in Retention Strategy: Heuristic Rules (RaaS) vs. Predictive Metrics (Ours)

- **RaaS Strategy (Hard-coded Rules):** To preserve Phoenix Tokens, RaaS opts to mandatorily retain all Prefill tokens without any eviction. For Decode tokens, it employs a timestamp-based LRU strategy.
- **LazyEviction Strategy (Dynamic Prediction):** We do not rely on the rigid assumption of "retaining all inputs." Instead, we leverage MRI (Maximum Recurrence Interval) to dynamically predict the future value of every token (whether Prefill or Decode). If a Prefill token shows no recurrence over a long period, LazyEviction evicts it.

3. Generalizability of Scenarios: RaaS is Limited by Input Length

- **Limitations of RaaS:** Since RaaS enforces the retention of all Prefill KV pairs, it fails in Long Context scenarios. The KV cache budget becomes saturated by the prefill tokens, leaving no space for decoding (as acknowledged in the RaaS limitations: "RaaS may lose crucial information... where the number of prefill tokens exceeds a certain threshold").
- **Advantages of LazyEviction:** Because we treat Prefill and Decode tokens equally for MRI-based compression, LazyEviction effectively handles Long-Context tasks (e.g., long-

document QA, long-code generation), maintaining high performance even under strict memory constraints.

H Checklist Issues

The datasets used in the experiment, including GSM8k (MIT), MATH500 (MIT), AIME (MIT), GPQA Dimond (MIT), Livecodebench (MIT) and models, including DeepSeek-R1-Distill-Llama-8B(MIT), DeepSeek-R1-Distill-Qwen-7B (MIT), Qwen3-4B (apache2.0), QwQ-32B (apache2.0), are used with their intended usage scenarios. We retrieve all models and datasets from HuggingFace[3], where detailed documentation, including parameter sizes and model architectures, is provided. We manually checked the data and believe there is no personal information misused.

We used ChatGPT for paper editing (e.g., grammar, spelling, word choice).

To the best of our knowledge, we believe our work does not pose risks that harm any subgroup of our society.

[3]https://huggingface.co/