

# RaaS: Reasoning-Aware Attention Sparsity for Efficient LLM Reasoning

Junhao Hu<sup>12\*</sup>, Wenrui Huang<sup>3</sup>, Weidong Wang<sup>3</sup>, Zhenwen Li<sup>1</sup>, Tiancheng Hu<sup>1</sup>,  
Zhixia Liu<sup>4</sup>, Xusheng Chen<sup>4</sup>, Tao Xie<sup>21†</sup>, Yizhou Shan<sup>4</sup>

<sup>1</sup>SCS, Peking University, Beijing, China

<sup>2</sup>Key Lab of HCST (PKU), MOE, Beijing, China

<sup>3</sup>School of Computer Science, Nanjing University, Nanjing, China

<sup>4</sup>Huawei Cloud, Shanghai, China

## Abstract

Large Language Models (LLMs) have demonstrated strong capabilities across various domains, with recent advancements in challenging reasoning tasks such as mathematics and programming. However, solving reasoning tasks often requires an LLM to generate long sequences, incurring  $O(N)$  time and memory complexities per token, where  $N$  is the current sequence length. To reduce complexities, existing sparsity-based algorithms propose to retain Key-Value (KV) vectors, the intermediate representations of only the most critical tokens. However, these algorithms struggle with the “impossible trinity” of accuracy, time, and memory. For example, the state-of-the-art algorithm, Quest, achieves high accuracy with  $O(L)$  time but  $O(N)$  memory ( $L$  is the cache budget,  $L \ll N$ ). To address the “impossible trinity”, in this paper, we identify a new attention pattern during the decode stage of reasoning tasks, where milestone tokens (analogous to lemmas in mathematical proofs) emerge, are utilized, and then become unimportant afterward. Based on this pattern, we propose a new algorithm *RaaS* that identifies milestone tokens and retains their KV vectors until they are no longer needed, achieving high accuracy with  $O(L)$  time and  $O(L)$  memory complexities.

## 1 Introduction

Large Language Models (LLMs) have gained widespread adoption due to their exceptional performance and versatility across various applications. However, their large-scale deployment faces a major obstacle: the high computational cost of long-sequence inference, which is increasingly common in modern user requests (i.e., prompts). This cost arises from the  $O(N)$  time and  $O(N)$  memory complexities required to generate each token, where  $N$  denotes the current sequence length

(i.e., input plus output tokens). Consequently, completing an entire request incurs a total time complexity of  $O(N^2)$ . For instance, the Llama 3.1 8B model supports sequences up to 128,000 tokens, leading to Job Completion Times (JCT) of several thousand seconds and memory usage up to 16GB per request<sup>1</sup>.

To study long-sequence inference, prior work divides LLMs’ generation process into two stages: prefill and decode. First, in the **prefill** stage, the model processes the prompt tokens given by users. It computes the Key (K) and Value (V) vectors for all prompt tokens, stores these vectors in the KV cache, and generates the first output token to initiate the decode stage. We collectively refer to prompt or input tokens as prefill tokens. Second, in the **decode** stage, the model iteratively processes each newly generated token. It computes the KV vectors for the new token, appends these vectors to the KV cache, and generates the next token. This process repeats until a specified stopping criterion is met. We refer to output tokens as decode tokens.

This paper focuses on optimizing the decode stage for two main reasons. First, long-decode tasks (producing long model outputs) have recently gained prominence, particularly in reasoning applications, as demonstrated by models such as OpenAI’s o1/o3 (OpenAI) and DeepSeek R1 (Dai et al., 2024). Despite their growing importance, the optimization of long-decode tasks remains under-explored compared to long-prefill tasks (containing long prompts) (Hu et al., 2024; Zheng et al., 2024; Kwon et al., 2023; Jin et al., 2024; Bai et al., 2024), such as Retrieval-Augmented Generation (RAG), few-shot learning, and tool use. Second, the decode stage represents a significant performance bottleneck in long-decode tasks. In reasoning applications, for instance, the time spent in the decode stage accounts for 99% of the JCT (Figure 1).

<sup>\*</sup>This work was completed during his internship at Huawei.

<sup>†</sup>Corresponding author.

<sup>1</sup><https://huggingface.co/blog/llama31>

Existing sparsity-based algorithms (Tang et al., 2024; Zhang et al., 2023; Xiao et al., 2024) optimize long-decode inference by retaining the KV vectors of only the most critical tokens, but struggle with the “impossible trinity” of accuracy, time, and memory (Figures 2 (b)(c)(d)). First, H2O, the pioneering work on attention sparsity, theoretically achieves  $O(L)$  time and memory complexities, where  $L$  indicates cache size and  $L \ll N$ . However, its inability to utilize efficient attention kernels and the lack of page-level KV management make it impractical. Moreover, H2O suffers from low accuracy. Second, StreamingLLM or Sink (Xiao et al., 2024) similarly offers  $O(L)$  time and memory complexities, but adopts an aggressive sparsification strategy that leads to extremely low accuracy on reasoning and other tasks (Tang et al., 2024)). Third, Quest (Tang et al., 2024), the state-of-the-art, achieves high accuracy with  $O(L)$  time complexity but  $O(N)$  memory complexity.

To maintain accuracy and  $O(L)$  time and memory complexities at the same time for reasoning tasks, we analyze their attention pattern during the decode stage, uncovering two key characteristics. First, we identify **milestone tokens**, which initially exhibit high attention scores but gradually receive lower scores and never receive high scores again. Analogous to lemmas in mathematical proofs, milestone tokens emerge, are utilized, and then fade away. These tokens, visible as bright columns (on the attention map) that slowly diminish, must be carefully managed to prevent significant accuracy loss (Figure 5). Second, we identify **phoenix tokens**, which receive low attention scores for a period long enough to be evicted from the cache but later regain importance. These tokens typically appear in the prefill tokens, such as user queries. Quest (Tang et al., 2024) retains the entire KV cache to avoid losing phoenix tokens, resulting in its  $O(N)$  memory complexity.

Based on the preceding observations, we propose a new *RaaS* algorithm that addresses the “impossible trinity” and consists of two main ideas. First, we identify milestone tokens and retain their KV vectors using a Least-Recently-Used (LRU) caching strategy. During each decoding step, tokens that receive attention scores above the median are considered **used** and are assigned the latest timestamp. Milestone tokens typically continue to receive the latest timestamps until they become permanently irrelevant. When the KV cache is full, *RaaS* evicts KV vectors of the tokens with

the oldest timestamp. Second, we retain the KV vectors of all prefill tokens without eviction. Since the phoenix tokens almost always appear within them in reasoning tasks, retaining these tokens’ KV vectors ensures that critical information is not lost during the decode stage.

We implement *RaaS* with 2k lines of Python code. To evaluate its performance, we compare it against H2O (Zhang et al., 2023), StreamingLLM (Xiao et al., 2024), and Quest (Tang et al., 2024) using three mathematical datasets on four reasoning-enabled models. Our experimental results demonstrate that *RaaS* achieves comparable accuracy and latency to Quest, while offering a significant advantage in memory efficiency ( $O(L)$  memory complexity). The code is available at: <https://github.com/DerekHJH/raas>.

In this paper, we make the following three main contributions:

- We identify a novel attention pattern in reasoning tasks, where milestone tokens (analogous to mathematical lemmas) emerge, are utilized, and then become unimportant.
- Based on the milestone pattern, we propose a new algorithm *RaaS* that achieves high accuracy with  $O(L)$  time and  $O(L)$  memory complexities.
- We implement and evaluate *RaaS*, demonstrating constant memory usage while maintaining similar accuracy and time performance compared to the state-of-the-art Quest.

## 2 Background and Motivation

In this section, we overview the Large Language Model (LLM) inference, highlighting the key concepts and challenges that motivate our work.

### 2.1 Autoregressive Generation and KV Cache

The generation process of LLMs consists of two distinct stages: the prefill stage and the decode stage (Hu et al., 2025b; Kwon et al., 2023). In the **prefill** stage, the model processes a sequence of prompt tokens all at once. It computes the Key (K) and Value (V) vectors for all prompt tokens, stores these vectors in the KV cache, and generates the first output token to initiate the decode stage. In the **decode** stage, the model iteratively processes each newly generated token. It computes the KV vectors for the new token, appends these vectors to the KV

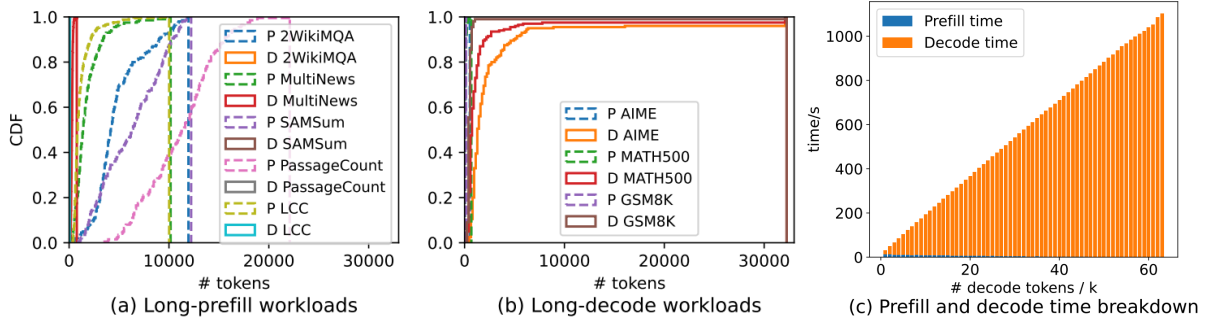


Figure 1: The Cumulative Distribution Function (CDF) of sequence lengths for the Prefill (P) and Decode (D) stages for (a) five datasets from LongBench (Bai et al., 2024) and (b) three math datasets running on the reasoning-enabled Marco-O1 model. (c) The breakdown of prefill and decode time during the inference of fixed 32k tokens using vLLM 0.6.1 with the LLaMA 3.1 8B model in FP16 precision. As the number of decode tokens increases (with the number of prefill tokens being 32k minus the decode tokens), the decode time rises significantly faster than the prefill time.

cache, and generates the next token. This process repeats until a specified stopping criterion is met. The KV cache (Pope et al., 2023; Hu et al., 2025a) accelerates the decode stage by allowing LLMs to process only the new token instead of reprocessing the entire sequence. With the KV cache, the attention mechanism incurs a time complexity of  $O(N)$  per decoding step and a memory complexity of  $O(N)$  for storing the KV cache, where  $N$  is the sequence length.

## 2.2 Cost Transfer: From Long-Prefill to Long-Decode Inference

Long-sequence inference incurs significant costs due to both memory and time requirements. First, it demands substantial memory resources, reaching up to 16 GB KV cache (in addition to the 16 GB model parameters) for processing 128k tokens running the LLaMA 3.1 8B model in FP16 precision<sup>2</sup>. Second, it requires considerable processing time, with inference for 32k tokens taking around 20 - 1000 seconds on vLLM 0.6.1 using the same model (Figure 1 (c)).

Long-sequence inference can be categorized into two types: long prefill and long decode. **Long prefill** arises from extensive input prompts, as observed in prior studies such as Retrieval-Augmented Generation (RAG) (Li et al., 2022; Jin et al., 2024; Gao et al., 2023; Jeong et al., 2024; Ram et al., 2023; Mao et al., 2021) (Figure 1 (a)). **Long decode** occurs particularly in reasoning-intensive tasks. Recent advancements emphasize reasoning, where models are guided to think, introspect, and iteratively refine their outputs (OpenAI;

Wang et al., 2024; Lightman et al., 2024; Zhao et al., 2024; Wei et al., 2022). This approach significantly enhances accuracy but shifts the computational burden to the decode stage. For instance, the OpenAI o1 model (OpenAI) requires approximately tens or hundreds of seconds<sup>3</sup> of “thinking time” before producing its final output. Given the prolonged decoding time and its already substantial proportion of the overall inference process (Figure 1 (b)), it is critically important to further optimize the decode stage to reduce both time and memory complexities.

## 2.3 Existing Sparsity-Based Algorithms

To reduce time and memory complexities of long-decode inference, one line of research uses sparsity-based algorithms (Xiao et al., 2024; Zhang et al., 2023; Tang et al., 2024; Chen et al., 2024) that retain the KV vectors of only the most critical tokens (fewer than 10% (Tang et al., 2024)). But these algorithms struggle with the “impossible trinity” of accuracy, time, and memory (Figure 2 (b)(c)(d)).

Figure 2 compares existing sparsity-based algorithms. First, the Dense or the standard attention algorithm (Vaswani et al., 2017) caches and reuses KV vectors of all tokens, achieving the highest accuracy but incurring  $O(N)$  time and memory complexities. Second, H2O (Zhang et al., 2023), the pioneering sparsity-based algorithm, caches and reuses KV vectors of recent tokens and important non-recent tokens. When the cache is full, it evicts non-recent tokens with the lowest accumulated attention scores. Although H2O theoret-

<sup>2</sup><https://huggingface.co/blog/llama31>

<sup>3</sup>[https://www.reddit.com/r/OpenAI/comments/1frdwqk/your\\_longest\\_thinking\\_time\\_gpt4\\_o1\\_mini/](https://www.reddit.com/r/OpenAI/comments/1frdwqk/your_longest_thinking_time_gpt4_o1_mini/)

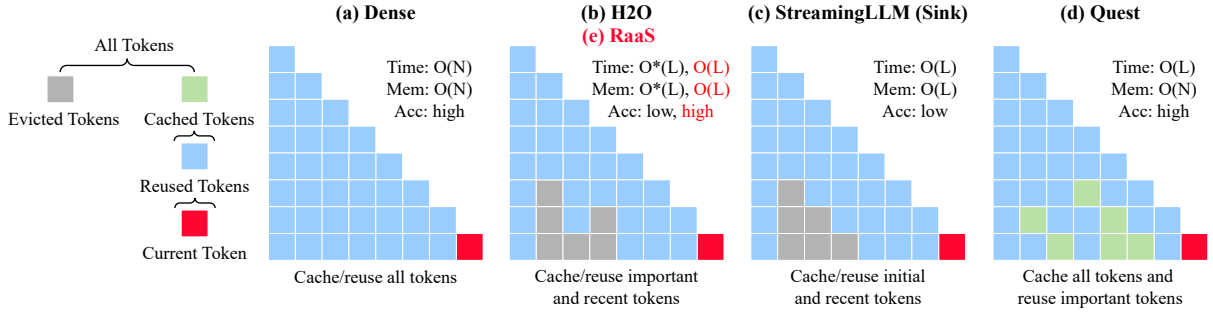


Figure 2: Comparison of sparsity-based algorithms.  $N$  indicates the sequence length while  $L$  indicates the cache budget where  $L \ll N$ . Asterisks on H2O’s time and memory complexities indicate theoretical complexities that are not realized in practical implementations. *RaaS* addresses the “impossible trinity” by achieving  $O(L)$  complexity for both time and memory, with accuracy comparable to Dense on reasoning tasks. Refer to Section 2 for detailed explanations of each algorithm’s design.

ically achieves  $O(L)$  time and memory complexities, where  $L \ll N$  denotes the cache budget, it suffers from low accuracy. Moreover, its lack of support for efficient attention kernels and page-level KV management limits its practical utility. Third, StreamingLLM or Sink (Xiao et al., 2024) statically decides to cache and reuse KV vectors of only the initial and recent tokens, without dynamically selecting important ones based on attention scores as H2O. Sink also offers  $O(L)$  time and memory complexities, but performs poorly on both reasoning and other tasks (Tang et al., 2024). Fourth, Quest caches KV vectors of all tokens but reuses only the ones with the top-k attention scores. Quest achieves high accuracy and  $O(L)$  time complexity but retains an  $O(N)$  memory complexity due to conservative caching.

### 3 Algorithm Design

To break the “impossible trinity” of sparsity-based algorithms on reasoning tasks, we analyze their decode stage and discover a new attention pattern (Section 3.1), based on which we design a new algorithm *RaaS* (Section 3.2) that achieves  $O(L)$  time and memory complexities, with accuracy comparable to Quest.

#### 3.1 Reasoning Attention Pattern

By analyzing the attention map of reasoning tasks’ decode stage, we discover two key characteristics (Figure 3). First, we identify **milestone tokens**, which initially exhibit high attention scores but gradually receive lower scores and never receive high scores again. Analogous to lemmas in mathematical proofs, milestone tokens emerge, are utilized, and then fade away. These tokens, visible as bright columns (on the attention map) that slowly

diminish (Figure 3 (a)), must be carefully managed to prevent significant accuracy loss (Figure 5). Second, we identify **phoenix tokens**, which receive low attention scores for a period long enough to be evicted from the cache but later regain importance. These tokens typically appear in short pre-fill prompts, such as user queries (Figure 3 (b)). Quest (Tang et al., 2024) retains the entire KV cache to avoid losing phoenix tokens, resulting in its  $O(N)$  memory complexity.

We offer a possible explanation for the milestone pattern in reasoning tasks. First, the emergence of milestone tokens is analogous to lemmas in mathematical proofs or subconclusions in thinking steps. Once an LLM generates milestone tokens, subsequent tokens primarily attend to the milestone tokens rather than the preceding tokens arriving at the milestone tokens. Second, the fading attention score of a milestone token mirrors the progression in mathematical reasoning. As reasoning advances from lower-level lemmas to higher-level ones, subsequent steps rely on the new lemmas rather than revisiting the older ones.

To illustrate the preceding explanation, consider one example<sup>4</sup> in Figure 4. First, tokens ①②③ serve as initial lemmas, which are crucial for subsequent deductions, corresponding to ①②③ columns in Figure 3 (a). Second, tokens ④⑤ serve as a new lemma, built upon ①②③, while at the same time, tokens ①②③ fade. Third, the final answer (token ⑥) attend to only the latest milestone tokens ④⑤.

On the other hand, the definition of phoenix tokens depends on the cache budget: a token qualifies as a phoenix token if it is evicted from the

<sup>4</sup>Examples abound during the investigation of reasoning tasks, not limited to this one, and not limited to those extra examples in the appendix.



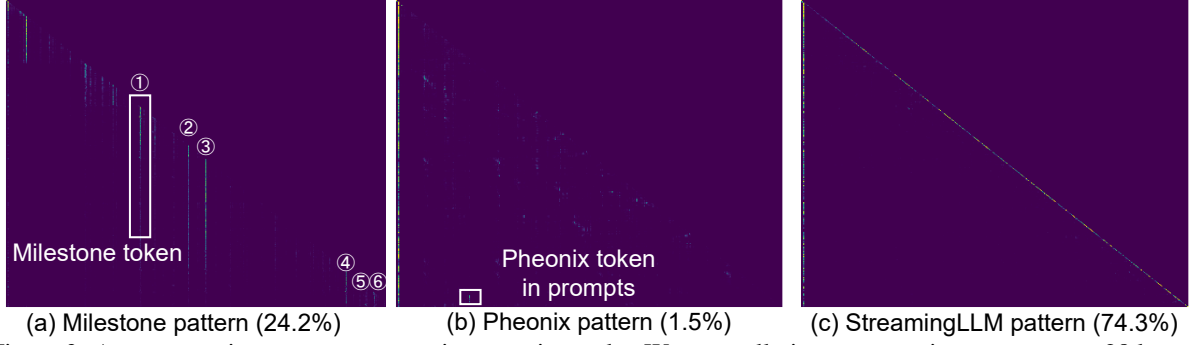


Figure 3: A new attention pattern emerges in reasoning tasks. We manually inspect attention maps across 28 layers and 28 heads of Qwen2.5-Math-7B-Instruct (Yang et al., 2024) on 100 MATH500 (Hendrycks et al., 2021) test cases. We find that (a) 24.2% maps with milestone tokens, (b) 1.5% maps with phoenix tokens (with a 64-token cache budget), (c) more than 70% “lazy” (Zhang et al., 2022) maps with StreamingLLM pattern. We use our best effort to balance the clarity and completeness of long-decode attention maps.

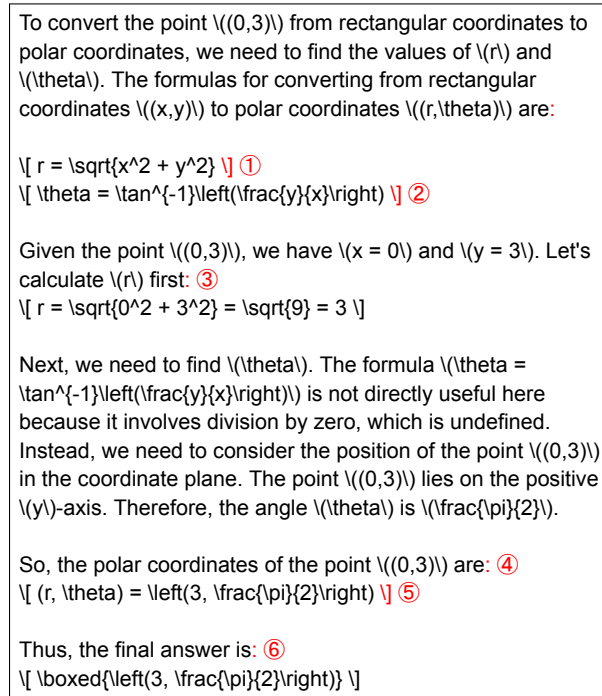


Figure 4: We input the prefill tokens, “...Convert the point (0, 3) to polar coordinates...”, to Qwen2.5-Math-7B-Instruct and obtain the corresponding decode tokens in the figure. The red tokens represent the milestone tokens or bright columns in Figure 3 (a).

cache and later reused. Under this definition, any token—including milestone tokens—can become a phoenix token. For instance, a milestone token that temporarily receives low attention scores (e.g., for four decoding steps) may be evicted under a cache budget of 4, thus becoming a phoenix token.

Phoenix tokens exhibit distinct behaviors in prefill and decode tokens. First, phoenix tokens rarely appear in decode tokens when using a sufficiently large cache budget. This behavior arises because decode tokens primarily consist of milestone to-

kens and other low-importance tokens, and milestone tokens are considered permanently irrelevant if they remain unused for an extended period. A sufficiently large cache budget prevents milestone tokens from becoming phoenix tokens. For example, with a cache budget of 512 tokens, phoenix tokens are rarely observed in decode tokens (Figure 5). In contrast, with a smaller budget of 64 tokens, phoenix tokens still occur, leading to degraded accuracy for *RaaS*. Second, phoenix tokens frequently occur in prefill tokens regardless of the cache budget. This behavior arises because LLMs typically refer back to the front user queries in the final conclusions of reasoning. To prevent the loss of critical query information, we retain KV vectors of prefill tokens and apply *RaaS* (Section 3.2) exclusively to decode tokens.

### 3.2 Design of *RaaS*

Based on the preceding observations, we propose a new *RaaS* algorithm that addresses the “impossible trinity” and consists of two main ideas. First, we identify milestone tokens and retain their KV vectors using a Least-Recently-Used (LRU) caching strategy. During each decoding step, tokens that receive attention scores above the median are considered **used** and are assigned the latest timestamp. Milestone tokens typically continue to receive the latest timestamps until they become permanently irrelevant. When the KV cache is full, *RaaS* evicts KV vectors of the tokens with the oldest timestamp. Second, we retain the KV vectors of all prefill tokens without eviction. Since prefill tokens are typically short and phoenix tokens almost always appear within them in reasoning tasks, retaining these tokens’ KV vectors ensures that critical infor-

---

**Algorithm 1** RaaS Algorithm

---

```
1: Input: A sequence X of prefill tokens, a model M, a KV manager kvm with  $r = 0.5$ 
2: Output: A sequence Y of prefill tokens plus decode tokens
3:  $Y = X$ 
4:  $y = M.forward(Y, \text{"prefill"})$ 
5: while  $y \neq \text{eos}$  and  $\text{len}(Y) < M.\text{context\_len}()$ 
6:    $Y.append(y)$ 
7:    $y = M.forward(Y, \text{"decode"})$ 
8: Return Y
9:
10: Function M.forward(X, stage)
11: ...
12:  $K\_old, V\_old = \text{kvm.load\_cache}(X)$ 
13: Generate Q, K_new, V_new for X's new tokens
14:  $K = K\_old::K\_new, V = V\_old::V\_new$ 
15:  $\text{score} = \text{attention}(Q, K, V)$ 
16:  $\text{kvm.store\_cache}(K, V, \text{score}, \text{stage})$ 
17: ...
18: End Function
19:
20: Function kvm.store_cache(K, V, score, stage)
21: if stage == "prefill"
22:   Store K, V and mark them as non-evictable
23: else if stage == "decode"
24:   Store K_new and V_new.
25:   New timestamps to KVs with top-r score
26:   if  $\text{kvm.is\_full}()$ 
27:     Evict KVs with the oldest timestamps
28: End Function
```

---

mation is not lost in the decode stage.

Algorithm 1 presents the detailed procedure of *RaaS*. (1) Given a sequence X of prefill tokens, the model M first performs the prefill stage, and then proceeds with the decode stage until a stopping criterion is met—such as the generation of an End-Of-Sequence (EOS) token or reaching the model’s maximum sequence length. (2) During each forward pass—whether in the prefill or decode stage—M loads the old KVs with a KV manager kvm, computes new KVs, concatenates them (denoted by “::”), calculates attention scores, and stores the new KVs. (3) The kvm’s store\_cache implements the core logic of *RaaS*, as described in the preceding paragraph. It retains a proportion  $r$  of tokens with the highest attention scores. Specifically, when  $r = 0.5$ , tokens with attention scores above the median are considered **used** and receive the latest timestamp.

The choice of  $r$  controls the distribution of tokens’ timestamps. If  $r$  is large, too many tokens will receive the latest timestamp, preventing effective differentiation of milestone tokens. Conversely, if  $r$  is small, most tokens are deemed irrelevant, potentially leading to the loss of milestone tokens. To address this dilemma, we propose to assign the latest timestamp to 50% ( $r = 0.5$ ) tokens with the highest attention scores in each decoding step, yielding good results (Figure 8).

### 3.3 Page-Based *RaaS*

Directly applying the version of *RaaS* in Section 3.2 faces two challenges. First, managing KV cache at the token level is inefficient, as small fragmentation in the cache complicates memory management and hinders efficient GPU computation. Second, *RaaS* requires the attention scores of all tokens to update timestamps, but retrieving these scores is incompatible with optimized attention kernels such as FlashAttention (Dao et al., 2022; Dao, 2024). As with H2O, bypassing fast kernels in favor of *RaaS* could result in degraded performance.

To address these challenges, we propose a page-based version of *RaaS*<sup>5</sup>. First, we introduce a page-based caching system with a fixed page size of  $\text{page\_size} = 16$  as in vLLM (Kwon et al., 2023). The timestamp management, as well as cache retention and eviction, is handled at the page level as in most of modern inference engines (Kwon et al., 2023; Zheng et al., 2024). Second, before using optimized attention kernels, we add a lightweight step to retrieve a representative attention score for each page to update its timestamp, similar to Quest. We select a representative Key (K) for each page, and the Query (Q) of the new decoding token attends to these representative keys to compute a single attention score per page. Based on these attention scores, we update the timestamp for each page and make eviction decisions at the page level. Various strategies exist for selecting a representative K, such as those used in Quest (Tang et al., 2024) and ArkVale (Chen et al., 2024). For fairness, we adopt the same representative selection strategy as in Quest.

## 4 Evaluation

We begin by describing the experimental setup, including implementation details, datasets, models,

---

<sup>5</sup>From now on, whenever we use *RaaS*, we refer to page-based *RaaS*.

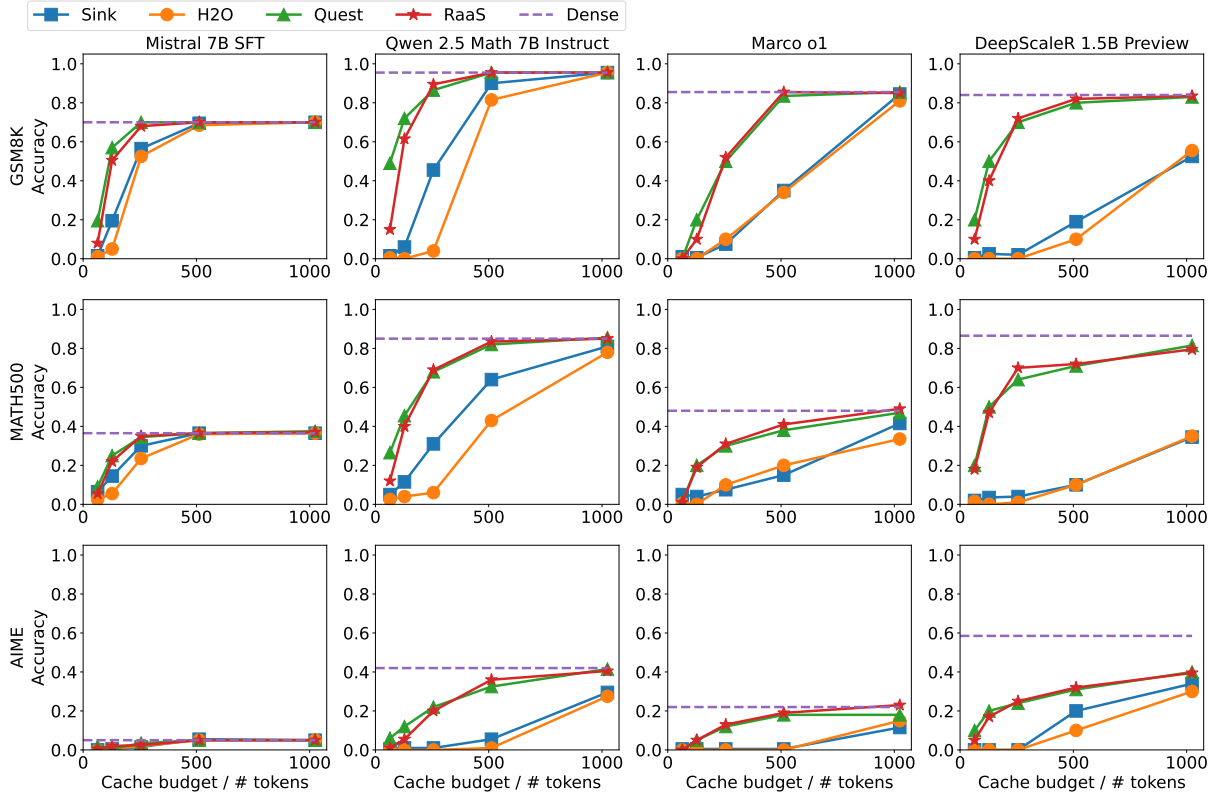


Figure 5: Accuracy vs. cache budget for five algorithms (legends) across three datasets (rows) and four models (columns). The y-axis shows the proportion of correctly solved problems among 200 test cases, while the x-axis represents varying cache budgets: 64, 128, 256, 512, and 1024.

evaluation metrics, and software/hardware environment. We then present key evaluation results.

#### 4.1 Experiment Setup

**Implementation.** We implement *RaaS* based on Hugging Face (Hugging Face) and Quest (Tang et al., 2024) with 2k lines of Python code. We port Quest from their public repository<sup>6</sup>.

**Datasets.** We take the first 200 test cases from each of the following three open-source datasets for our benchmarks: *GSM8K* (Cobbe et al., 2021), *MATH500* (Hendrycks et al., 2021), and *AIME* (AIME), to test the reasoning ability of language models. First, *GSM8k* (Cobbe et al., 2021) contains 8.5k high-quality, linguistically diverse grade-school math problems. These human-written problems need solutions that involve multi-step reasoning and a series of basic arithmetic operations. Second, *MATH500* (Hendrycks et al., 2021) contains 500 challenging problems sourced from high school math competitions with five distinct levels based on the Art of Problem Solving (AoPS) framework, ranging from level 1 to level 5. Third,

*AIME* (AIME) is a math problem dataset collected from the American Invitational Mathematics Examination (AIME) competition from 1983 to 2024, designed to challenge the most exceptional high school math students in the United States. These problems cover various fields, such as algebra, geometry, and number theory.

**Models.** We evaluate our algorithm using four popular models: *Marco-o1* (Zhao et al., 2024), *Qwen2.5-Math-7B-Instruct* (Wang et al., 2024), *Mistral-Math-7B* (Wang et al., 2024), and *DeepScaleR-1.5B*<sup>7</sup>. They are four of the most powerful open-source LLMs with long-reasoning capabilities.

**Metrics.** We use two metrics to evaluate performance and model accuracy. First, *Job Completion Time (JCT)* is the time from when users send a request (prompt) to LLMs to when users receive a complete response. A smaller *JCT* indicates a faster algorithm. Second, *Accuracy* (Wang et al., 2024) measures the mathematical equivalence between an LLM’s output and the ground-truth an-

<sup>6</sup><https://github.com/mit-han-lab/Quest>. Accessed on Oct 2024.

<sup>7</sup><https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303013a4e2>

swer. For each test case, it is either correct or incorrect, and the overall accuracy is reported as the percentage of correctly solved problems across the entire dataset.

**Baselines.** We compare *RaaS*’ accuracy with Dense, H2O, StreamingLLM, and Quest. We implement H2O and StreamingLLM using the HuggingFace Cache class. We compare *RaaS*’s latency and memory consumption with only Dense and Quest because StreamingLLM and H2O achieve too low accuracy to be included. We use Quest’s official repository with *page\_size* = 16.

**Environment.** We run experiments on a single NVIDIA A100 server with one A100-80GB GPU available. It has 128-core Intel(R) Xeon(R) Platinum 8358P CPU@2.60GHz with two hyperthreading and 1TB DRAM. We use Ubuntu 20.04 with Linux kernel 5.16.7 and CUDA 12.6. Unless stated otherwise, we set  $r = 0.5$  and *page\_size* = 16.

## 4.2 Accuracy and Cache Budget Trade-off

We evaluate five algorithms across three datasets and four models, yielding three key insights from the experimental results (Figure 5). First, H2O and Sink exhibit poor accuracy under fixed cache budgets compared to others. Sink indiscriminately discards important tokens, including milestone tokens. H2O, on the other hand, overemphasizes accumulated historical attention scores, leading it to retain outdated milestone tokens for too long while discarding newer, relevant ones. Second, Quest and *RaaS* achieve the best accuracy. Quest retains KV vectors of all tokens while *RaaS* optimizes memory usage by carefully handling milestone tokens with  $O(L)$  memory complexity (Figure 6). Across these datasets, a cache budget of 1024 tokens is generally sufficient to match Dense’s accuracy. Third, when the cache budget is small, *RaaS* underperforms because *RaaS* retains all prefill tokens, and with a limited cache budget, most of the budget is allocated to prefill tokens, causing almost all decoding tokens to be discarded, negatively impacting accuracy. For small cache budgets or long-prefill scenarios, we recommend using Quest for prefill tokens and *RaaS* for decode tokens.

## 4.3 Latency/Memory vs. Decoding Length

We evaluate the Dense, Quest, and *RaaS* in terms of their time and memory complexities, yielding two key observations from the experimental results (Figure 6). First, as the number of decode tokens increases, Dense’s JCT grows quadratically,

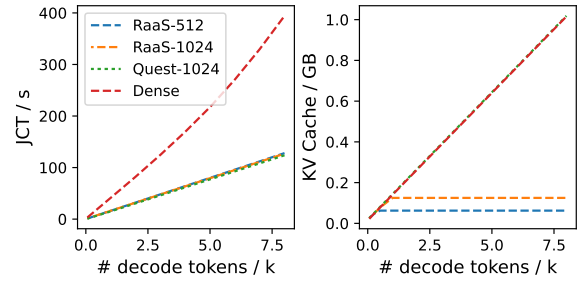


Figure 6: Latency and memory consumptions of Dense, Quest, and *RaaS* running on Mistral-Math-7B, using workloads with a fixed prefill length (128 tokens), varying decode lengths (from 0 to 8k tokens) and a batch size of 4. We use dashed lines to improve the visibility of overlapping lines.

while both *RaaS* and Quest exhibit linear latency growth. The reason is that Dense has  $O(N^2)$  time complexity, whereas *RaaS* and Quest have  $O(NL)$  time complexity, reducing each decoding step from  $O(N)$  to  $O(L)$ . Second, as the number of decode tokens increases, the memory consumption of Dense and Quest grows linearly, while *RaaS* initially increases linearly but plateaus once the number of decode tokens exceeds its cache budget. The reason is that Dense and Quest have  $O(N)$  memory complexity, whereas *RaaS* achieves  $O(L)$  memory complexity. With a smaller memory footprint, inference engines using *RaaS* are likely to achieve significantly higher throughput.

## 4.4 Micro-Benchmarks

**The impact of discarding milestone tokens.** Figure 7 shows that discarding milestone tokens, as in H2O-128 and Sink-128, increases the decode lengths. Sometimes, the decode length increases without solving the problem. Analysis of the outputs reveals that while the model initially reasons correctly for the first few tokens (e.g., green tokens in Figure 7), it loses track (orange tokens) of the reasoning process when milestone tokens are discarded, leading to repeated attempts at re-reasoning (red tokens), which ultimately results in the model getting stuck indefinitely.

**The impact of  $r$ .** The choice of  $r$  affects the distribution of tokens’ timestamps, with  $r = 0.5$  generally yielding optimal results, as shown in Figure 8. First, when  $r$  is small, too many tokens are assigned the latest timestamp, preventing effective differentiation of milestone tokens. Second, when  $r$  is big, most tokens are deemed irrelevant, potentially leading to the loss of milestone tokens.



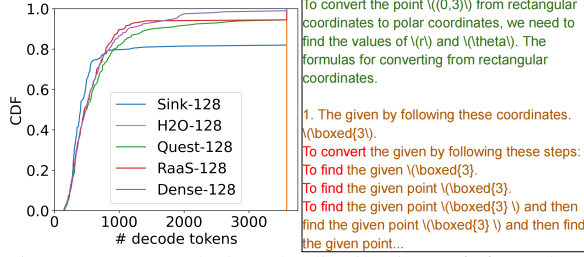


Figure 7: Decode-length distribution of five algorithms using Qwen2.5-Math-7B-Instruct with 4k context length, on MATH500. For example, H2O-128 (128-token cache) always generates to the 4k length limit without solving the problem. On the right, we show a decoding example of H2O-128.

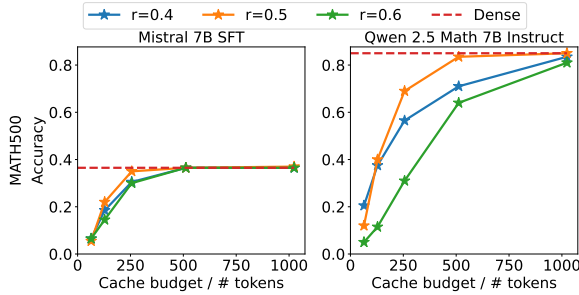


Figure 8: Accuracy of RaaS with different cache budgets and  $r$ s.

## 5 Related Work

Many approaches have been proposed to reduce the time and memory complexities of long-sequence inference; these approaches can be categorized into two types: one that modifies the model architecture and the other that is more plug-and-play.

### 5.1 Model Architecture

Two types of approaches have emerged for altering model architecture. First, some approaches modify the inner workings of the Transformer while retaining its overall structure. For example, Multi-Query Attention (MQA) (Shazeer, 2019) and Group-Query Attention (GQA) (Ainslie et al., 2023) reduce the number of KV heads, achieving similar accuracy to full-head configurations. Second, some approaches change the Transformer architecture significantly in favor of alternative paradigms. For example, RWKV (Peng et al., 2023), RetNet (Sun et al., 2023), and Mamba (Gu and Dao, 2023) adopt RNN-based models, offering lower computational costs but typically underperform compared to Transformer-based models.

### 5.2 KV Compression

Two primary types of KV compression have emerged: KV quantization and KV pruning. First, KV quantization approaches (Xiao et al., 2023; Yao et al., 2022; Dettmers et al., 2022; Liu et al., 2024) map higher precision KVs into lower ones, trading accuracy for savings in time and memory. Second, KV pruning approaches focus on leveraging attention sparsity (Zhang et al., 2023; Ge et al., 2024; Jiang et al., 2024; Cai et al., 2024; Fu et al., 2024; Xiao et al., 2025), which states that only a few tokens are crucial during LLM inference. Thus, evicting less important tokens from the KV cache is a key strategy for reducing time and memory. For example, StreamingLLM (Xiao et al., 2024) and LM-Infinite (Han et al., 2024) evict fixed-position tokens, retaining only the initial and recent window tokens. H2O (Zhang et al., 2023), SnapKV (Li et al., 2024), ScissorHands (Liu et al., 2023), and TOVA (Oren et al., 2024) keep the recent tokens and the top- $k$  important tokens based on the attention score calculated within a local window. More recent work, such as Quest (Tang et al., 2024) and ArkVale (Chen et al., 2024), manages the KV cache at the page level, selecting the top- $k$  important pages during each generation step to reduce time complexity.

Our work presents a new trial of applying KV pruning in reasoning tasks, which are characterized by a new milestone attention pattern. For the first time, we achieve real  $O(L)$  time and memory complexities with high accuracy.

## 6 Conclusion

In this paper, we have identified a new milestone attention pattern observed in the decode stage of reasoning tasks. Leveraging this pattern, we have proposed a sparsity-based algorithm *RaaS* that achieves high accuracy while maintaining  $O(L)$  time and  $O(L)$  memory complexities. Our experiments, conducted across three datasets and four reasoning-enabled models, demonstrate that *RaaS* delivers comparable accuracy and latency to the state-of-the-art Quest, but with constant memory consumption. The key to *RaaS*' success lies in the handling of milestone tokens, which represent intermediate conclusions leading to the final output.

### Limitations

Our work in this paper has the following major limitations.

**Lack of comprehensive attention map statistics.** Attention patterns vary across layers, heads, and model architectures. Manual inspection, without rigorous statistical analysis, is insufficient for drawing general conclusions about emerging attention patterns. Our study remains small-scale due to the absence of an automated tool for analyzing attention patterns across datasets, models, and layers. Such a tool would take an input defining an attention pattern, a range of datasets and models, and output the statistics of the specified attention pattern across each model’s layers and heads. Although we cannot address this limitation for now, we plan to develop such a tool in the future, and this tool would benefit the entire research community.

**Limited applicability of *RaaS*.** *RaaS* is specifically designed for reasoning tasks where the number of prefill tokens (e.g., a mathematical query) is small but the number of decode tokens (e.g., a chain of reasoning followed by a final answer) is large. Given that *RaaS* retains KV vectors of all prefill tokens, it allocates the entire cache budget to them and discards nearly all decode tokens in long-prefill scenarios. Therefore, in these scenarios, we recommend using the combination of Quest (on only prefill tokens) and *RaaS* (on only decode tokens).

**Evaluation on a limited set of datasets and models.** Our evaluation covers only four models and three datasets. As such, the results may not generalize beyond these specific configurations. Although models with longer context lengths (e.g., Qwen2.5-Max, DeepSeek-r1) and datasets such as GPQA Diamond and Codeforces exist, exhaustive evaluation across all combinations is computationally prohibitive (Hu et al., 2023). As reported in prior work (Zhong et al., 2024), decoding a single token can take approximately 30 ms; thus, processing 16k tokens on an A100-80GB GPU requires around 8 minutes. Running 200 test cases would take over a day on a single GPU, making large-scale evaluation infeasible with limited resources. Nonetheless, we believe that the core idea of the milestone pattern and its underlying rationale—the thinking/reasoning process is step-by-step, constructing lemmas along the way—remains broadly applicable.

## Acknowledgments

This work was partially supported by National Natural Science Foundation of China under Grant No.

92464301. We would also like to thank the anonymous reviewers for their insightful comments and suggestions, which help improve the quality of this paper.

## References

- AIME. AIME. [https://huggingface.co/datasets/di-zhang-fdu/AIME\\_1983\\_2024](https://huggingface.co/datasets/di-zhang-fdu/AIME_1983_2024).
- Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901.
- Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. LongBench: A bilingual, multi-task benchmark for long context understanding. In *Proceedings of the Sixty-Second Annual Meeting of the Association for Computational Linguistics*, pages 3119–3137.
- 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. *CoRR*.
- Renze Chen, Zhuofeng Wang, Beiquan Cao, Tong Wu, Size Zheng, Xiuhong Li, Xuechao Wei, Shengen Yan, Meng Li, and Yun Liang. 2024. ArkVale: Efficient generative LLM inference with recallable key-value eviction. In *Proceedings of the Thirty-Eighth Annual Conference on Neural Information Processing Systems*, pages 113134–113155.
- 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. *CoRR*.
- Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. 2024. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the Sixty-Second Annual Meeting of the Association for Computational Linguistics*, pages 1280–1297.
- Tri Dao. 2024. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *Proceedings of the Twelfth International Conference on Learning Representations*.

- Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Proceedings of the Thirty-Sixth Annual Conference on Neural Information Processing Systems*, pages 16344–16359.
- Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In *Proceedings of the Thirty-Sixth Annual Conference on Neural Information Processing Systems*, pages 30318–30332.
- Qichen Fu, Minsik Cho, Thomas Merth, Sachin Mehta, Mohammad Rastegari, and Mahyar Najibi. 2024. LazyLLM: Dynamic token pruning for efficient long context LLM inference. *CoRR*.
- Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *CoRR*.
- Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2024. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *Proceedings of the Twelfth International Conference on Learning Representations*.
- Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *CoRR*.
- Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2024. LM-Infinite: Zero-shot extreme length generalization for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3991–4008.
- 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. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*.
- Cunchen Hu, Heyang Huang, Junhao Hu, Jiang Xu, Xusheng Chen, Tao Xie, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, and Yizhou Shan. 2024. MemServe: Context caching for disaggregated LLM serving with elastic memory pool. *CoRR*.
- Junhao Hu, Wenrui Huang, Haoyi Wang, Weidong Wang, Tiancheng Hu, Qin Zhang, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. 2025a. EPIC: efficient position-independent caching for serving large language models. In *Proceedings of the Forty-Second International Conference on Machine Learning*.
- Junhao Hu, Chaozheng Wang, Hailiang Huang, Huang Luo, Yu Jin, Yuetang Deng, and Tao Xie. 2023. Predicting compilation resources for adaptive build in an industrial setting. In *Proceedings of the Thirty-Eighth IEEE/ACM International Conference on Automated Software Engineering*, pages 1808–1813.
- Junhao Hu, Jiang Xu, Zhixia Liu, Yulong He, Yuetao Chen, Hao Xu, Jiang Liu, Baoquan Zhang, Shining Wan, Gengyuan Dan, Zhiyu Dong, Zhihao Ren, Jie Meng, Chao He, Changhong Liu, Tao Xie, Dayun Lin, Qin Zhang, Yue Yu, Hao Feng, Xusheng Chen, and Yizhou Shan. 2025b. DEEPSERVE: Serverless large language model serving at scale. In *Proceedings of the 2025 USENIX Annual Technical Conference*.
- Hugging Face. Hugging Face. <https://huggingface.co>.
- Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong Park. 2024. Adaptive-RAG: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 7036–7050.
- Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024. MInference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. In *Proceedings of the Thirty-Eighth Annual Conference on Neural Information Processing Systems*, pages 52481–52515.
- Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. 2024. RAGCache: Efficient knowledge caching for retrieval-augmented generation. *CoRR*.
- 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 Twenty-Ninth Symposium on Operating Systems Principles*, pages 611–626.
- Huayang Li, Yixuan Su, Deng Cai, Yan Wang, and Lemaou Liu. 2022. A survey on retrieval-augmented text generation. *CoRR*.
- 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. In *Proceedings of the Thirty-Eighth Annual Conference on Neural Information Processing Systems*, pages 22947–22970.
- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let’s verify step by step. In *Proceedings of the Twelfth International Conference on Learning Representations*.



- Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhao Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems*, pages 52342–52364.
- Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhao Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the Forty-First International Conference on Machine Learning*, pages 32332–32344.
- Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. 2021. Generation-augmented retrieval for open-domain question answering. In *Proceedings of the Fifty-Ninth Annual Meeting of the Association for Computational Linguistics and the Eleventh International Joint Conference on Natural Language Processing*, pages 4089–4100.
- OpenAI. OpenAI o1. <https://openai.com/o1/>.
- Matanel Oren, Michael Hassid, Yarden Nir, Yossi Adi, and Roy Schwartz. 2024. Transformers are multi-state RNNs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18724–18741.
- Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Leon Derczynski, Xingjian Du, Matteo Grella, Kranthi Kiran GV, Xuzheng He, Haowen Hou, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartłomiej Koptyra, Hayden Lau, Jiaju Lin, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Guangyu Song, Xiangru Tang, Johan S. Wind, Stanislaw Wozniak, Zhenyuan Zhang, Qinghua Zhou, Jian Zhu, and Rui-Jie Zhu. 2023. RWKV: Reinventing RNNs for the transformer era. In *Findings of the Association for Computational Linguistics: EMNLP*, pages 14048–14077.
- Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. In *Proceedings of the Sixth Conference on Machine Learning and Systems*, pages 606–624.
- Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlga, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. 2023. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*.
- Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *CoRR*.
- Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. 2023. Retentive network: A successor to transformer for large language models. *CoRR*.
- Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. QUEST: query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the Forty-First International Conference on Machine Learning*, pages 47901–47911.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the Thirty-First Annual Conference on Neural Information Processing Systems*, pages 5998–6008.
- Jun Wang, Meng Fang, Ziyu Wan, Muning Wen, Jiachen Zhu, Anjie Liu, Ziqin Gong, Yan Song, Lei Chen, Lionel M. Ni, Linyi Yang, Ying Wen, and Weinan Zhang. 2024. OpenR: An open source framework for advanced reasoning with large language models. *CoRR*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the Thirty-Sixth Annual Conference on Neural Information Processing Systems*, pages 24824–24837.
- Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the Fortieth International Conference on Machine Learning*, pages 38087–38099.
- Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. 2025. DuoAttention: Efficient long-context LLM inference with retrieval and streaming heads. In *Proceedings of the Thirteenth International Conference on Learning Representations*.
- Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *Proceedings of the Twelfth International Conference on Learning Representations*.
- An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. 2024. Qwen2.5-Math technical report: Toward mathematical expert model via self-improvement. *CoRR*.
- Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. ZeroQuant: Efficient and affordable post-training quantization for large-scale transformers. In *Proceedings of the Thirty-Sixth Annual Conference on Neural Information Processing Systems*, pages 27168–27183.



- Xuan Zhang, Fengzhuo Zhang, Cunxiao Du, Chao Du, Tianyu Pang, Wei Gao, and Min Lin. 2022. Light-Transfer: your long-context LLM is secretly a hybrid model with effortless adaptation. *CoRR*.
- Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2O: heavy-hitter oracle for efficient generative inference of large language models. In *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems*, pages 34661–34710.
- Yu Zhao, Huifeng Yin, Bo Zeng, Hao Wang, Tianqi Shi, Chenyang Lyu, Longyue Wang, Weihua Luo, and Kaifu Zhang. 2024. Marco-o1: Towards open reasoning models for open-ended solutions. *CoRR*.
- Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. 2024. SGLang: Efficient execution of structured language model programs. In *Proceedings of the Thirty-Eighth Annual Conference on Neural Information Processing Systems*, pages 62557–62583.
- Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *Proceedings of the Eighteenth USENIX Symposium on Operating Systems Design and Implementation*, pages 193–210.

Prompt: If  $f(x) = \frac{3x-2}{x-2}$ , what is the value of  $f(-2) + f(-1) + f(0)$ ? Express your answer as a common fraction.

Decode:

To find the value of  $f(-2) + f(-1) + f(0)$  for the function  $f(x) = \frac{3x-2}{x-2}$ , we need to evaluate  $f(x)$  at  $x = -2$ ,  $x = -1$ , and  $x = 0$  separately and then sum the results. ①

First, let's evaluate  $f(-2)$ :

$$f(-2) = \frac{3(-2) - 2}{-2 - 2} = \frac{-6 - 2}{-4} = \frac{-8}{-4} = 2$$
 ②

Next, let's evaluate  $f(-1)$ :

$$f(-1) = \frac{3(-1) - 2}{-1 - 2} = \frac{-3 - 2}{-3} = \frac{-5}{-3} = \frac{5}{3}$$
 ③

Then, let's evaluate  $f(0)$ :

$$f(0) = \frac{3(0) - 2}{0 - 2} = \frac{-2}{-2} = 1$$
 ④

Now, we sum the values of  $f(-2)$ ,  $f(-1)$ , and  $f(0)$ :

$$f(-2) + f(-1) + f(0) = 2 + \frac{5}{3} + 1$$
 ⑤

To add these numbers, we need a common denominator. The common denominator for 2,  $\frac{5}{3}$ , and 1 is 3. We convert 2 and 1 to fractions with a denominator of 3:

$$2 = \frac{6}{3} \quad \text{and} \quad 1 = \frac{3}{3}$$
 ⑥

Now we can add the fractions:

$$\frac{6}{3} + \frac{5}{3} + \frac{3}{3} = \frac{6 + 5 + 3}{3} = \frac{14}{3}$$
 ⑦

Therefore, the value of  $f(-2) + f(-1) + f(0)$  is  $\boxed{\frac{14}{3}}$ .

Figure 10: Milestone example.

Prompt: Define  $p = \sum_{k=1}^{\infty} \frac{1}{k^2}$  and  $q = \sum_{k=1}^{\infty} \frac{1}{k^3}$ . Find a way to write  $\sum_{j=1}^{\infty} \sum_{k=1}^{\infty} \frac{1}{(j+k)^3}$  in terms of  $p$  and  $q$ .

Decode

To find a way to write the double sum  $\sum_{j=1}^{\infty} \sum_{k=1}^{\infty} \frac{1}{(j+k)^3}$  in terms of  $p$  and  $q$ , we start by considering the sum in a different form. We can rewrite the double sum by changing the order of summation. Instead of summing over  $j$  and  $k$  directly, we sum over  $n = j+k$  and then over the possible values of  $j$  and  $k$  that satisfy  $j+k=n$ . ①

The sum  $\sum_{j=1}^{\infty} \sum_{k=1}^{\infty} \frac{1}{(j+k)^3}$  can be rewritten as:

$$\sum_{n=2}^{\infty} \sum_{j=1}^{n-1} \frac{1}{n^3}$$
 ②

Here, the inner sum  $\sum_{j=1}^{n-1} \frac{1}{n^3}$  is simply  $\frac{n-1}{n^3}$  because there are  $(n-1)$  terms in the sum, each equal to  $\frac{1}{n^3}$ . Therefore, the double sum becomes:

$$\sum_{n=2}^{\infty} \frac{n-1}{n^3} = \sum_{n=2}^{\infty} \left( \frac{1}{n^2} - \frac{1}{n^3} \right)$$
 ③

We can split this into two separate sums:

$$\sum_{n=2}^{\infty} \frac{1}{n^2} - \sum_{n=2}^{\infty} \frac{1}{n^3}$$
 ④

The first sum  $\sum_{n=2}^{\infty} \frac{1}{n^2}$  is the same as  $(p - 1)$  because  $p = \sum_{n=1}^{\infty} \frac{1}{n^2}$ . The second sum  $\sum_{n=2}^{\infty} \frac{1}{n^3}$  is the same as  $(q - 1)$  because  $q = \sum_{n=1}^{\infty} \frac{1}{n^3}$ . Therefore, the expression becomes:

$$(p - 1) - (q - 1) = p - q$$
 ⑤

Thus, the double sum  $\sum_{j=1}^{\infty} \sum_{k=1}^{\infty} \frac{1}{(j+k)^3}$  in terms of  $p$  and  $q$  is:  $\boxed{p - q}$

Figure 9: Milestone example.

## A More Examples for Milestone Tokens

This section presents more examples of milestone tokens in Figure 9 and Figure 10. For all examples, we input the prefill tokens to Qwen2.5-Math-7B-Instruct and obtain the corresponding decode tokens, as shown in the figure. The tokens marked red represent the milestone tokens. Although we only show a few examples here, the milestone patterns abound in reasoning tasks.

## B Checklist-Related Issues

Three datasets GSM8k (MIT), MATH500 (MIT), AIME (MIT), and four models Mistral Math 7B (No licence), Qwen 2.5 Math 7B Instruct (apache-2.0), Marco o1 (apache-2.0), DeepScaleR 1.5B Preview (MIT) are used with their intended usage scenarios. We retrieve all models and datasets from Hugging Face, 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 to check the grammar of the texts.

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