RaaS: Reasoning-Aware Attention Sparsity for Efficient LLM Inference with Long Decoding Chains

Anonymous ACL submission

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 long decoding chains (of thoughts), which incur $O(N)$ time and memory consumption, where N is the chain length. To mitigate $O(N)$ time and memory consumption, existing sparsity-based algorithms propose retaining only the most critical token's intermediate data (i.e., key-value cache) and discarding the rest. However, these existing 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 this issue, 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 named *RaaS* that identifies and retains milestone tokens only until they are no longer needed, achieving high accuracy with $O(L)$ time and $O(L)$ memory complexity.

1 Introduction

Large Language Models (LLMs) have gained widespread adoption due to their exceptional performance and versatility across various applications. However, a significant challenge to large-scale deployment and application is the high computational cost associated with long-context inference. LLMs must process an entire prompt during the prefill stage and then generate tokens autoregressively during the decode stage. Both stages require significant processing time and memory for intermediate data, specifically the Key-Value (KV) cache. For standard attention algorithms (also referred to as Dense (Chen et al.) algorithms), the time and memory complexity is $O(N)$, where N is the sequence length. For example, in the Llama 3.1 8B model, sequences can grow up to 128k tokens, resulting in potentially thousands of seconds of processing time and 16GB of KV cache for a single request.

There are two primary types of long-context inference. The first type is long-prefill inference, commonly encountered in Retrieval-Augmented Generation (RAG) tasks, where the used LLM is required to process a lengthy prompt before generating responses. Previous research (Hu et al., 2024; Zheng et al., 2024; Kwon et al., 2023; Jin et al., 2024; Bai et al., 2024) has primarily focused on this inference type. The second type is long-decode inference, which has recently gained prominence in reasoning tasks, such as those exemplified by OpenAI's models (OpenAI) (e.g., o1, o3) and DeepSeek R1 (Dai et al., 2024). In reasoning tasks, the decode stage accounts for 99% of the Job-Completion-Time (JCT) (Figure 1), becoming a critical bottleneck.

To mitigate high time and memory consumption in long-prefill scenarios, existing sparsity-based algorithms (Tang et al., 2024; Zhang et al., 2023; Xiao et al., 2024b) propose retaining only the most critical token's KV and discarding the rest. However, when directly applied in long-decode scenarios, these existing algorithms struggle with the "impossible trinity" of accuracy, time, and memory (Figures 2 (b)(c)(d)). For example, the state-of-the-art algorithm, Quest (Tang et al., 2024), achieves high accuracy with $O(L)$ time but $O(N)$ memory, where L is the cache budget and $L \ll N$.

To achieve high accuracy and $O(L)$ time/memory complexity at the same time, we analyze the attention pattern during the decode stage of reasoning tasks, 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—similar to a water column in a waterfall (Figure 3)—must be carefully managed to prevent significant accuracy loss (Figure 6). 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 prefill prompts, such as mathematical questions. Quest (Tang et al., 2024) retains the entire KV cache to avoid losing phoenix tokens, resulting in the $O(N)$ memory complexity.

Based on the preceding observations, we propose a simple yet effective algorithm named *RaaS* that addresses the "impossible trinity" and consists of two main ideas. First, we identify and retain milestone tokens only until they are no longer needed, using timestamps to track their importance. When a token receives an attention score above α (e.g., $\alpha = 0.0001$), we assign it the latest timestamp. Milestone tokens always receive the latest timestamp until it becomes unimportant. When the cache is full, we evict tokens with the oldest timestamp. Second, we retain the KV cache 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 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., 2024b), 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).

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

- We identify a novel waterfall attention pattern in reasoning tasks, where milestone tokens (analogous to mathematical lemmas) emerge, are utilized, and then become unimportant.
- Based on the waterfall attention pattern, we propose a new algorithm *RaaS* that achieves high accuracy with $O(L)$ time and $O(L)$ memory complexity.
- We implement *RaaS* in a system, 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 & KV Cache

LLMs generate tokens autoregressively, predicting one token at a time based on the input. This process involves two stages: the prefill stage and the decode stage. In the **prefill** stage, LLMs process the entire input prompt $(x_1, x_2, \ldots, x_n)$, computing and caching the key and value vectors for each token. This stage can be slow for long inputs, and the time to generate the first token is measured by the Time-to-First-Token (TTFT) metric. In the **decode** stage, LLMs generate one token at a time. The model computes the probability of the next token x_{n+1}, selects the most likely token, and appends its key and value vectors to the KV cache.

The KV cache (Pope et al., 2023), which stores the key and value vectors computed by the attention module, accelerates generation by allowing the model to process only the new token instead of recalculating KV for the entire sequence. With the KV cache, the attention mechanism exhibits a computational complexity of $O(N)$ for one decoding step and a memory complexity of $O(N)$ for storing the KV cache, where N is the number of tokens or the sequence length.

2.2 Cost Transfer: from Long-Prefill to Long-Decode Inference

Long-context 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[1]. 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)).

[1] https://huggingface.co/blog/llama31

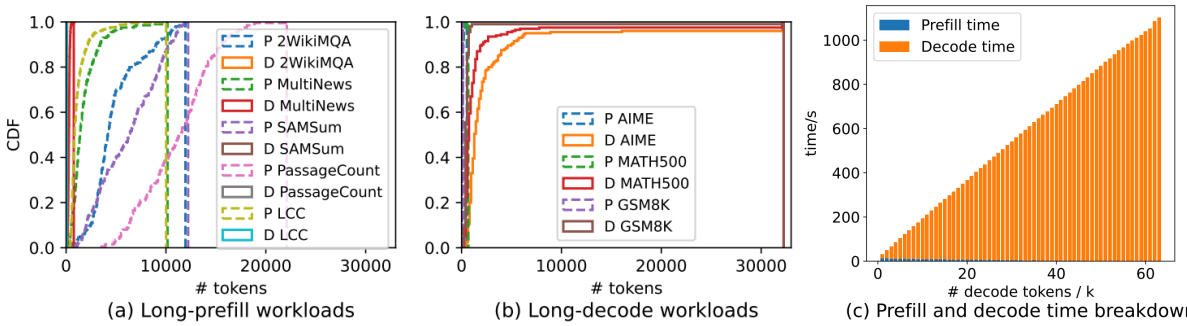


Figure 1: The Cumulative Distribution Function (CDF) of token lengths for the prefill (P, in dotted lines) and decode (D, in solid lines) phases for (a) five datasets from LongBench (Bai et al., 2024) and (b) three math datasets running on the reasoning-enabled Marco-O1 model. (c) The breakdown of prefill 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 (orange bars) rises significantly faster than the prefill time (blue bars).

Long-context 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[2] 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 latency and memory consumption.

2.3 Existing Sparsity-Based Algorithms

To reduce memory and time complexity in long-prefill scenarios, one line of research proposes sparsity-based algorithms (Xiao et al., 2024b; Zhang et al., 2023; Tang et al., 2024; Chen et al.). Sparsity-based algorithms propose retaining only the most critical tokens' (fewer than 10% (Tang et al., 2024)) KV and discarding the rest. However, when directly applied in long-decode scenarios, existing algorithms struggle with the "impossible trinity" of accuracy, time, and memory (Figure 2 (b)(c)(d)).

The differences among existing algorithms are shown in Figure 2. First, the Dense or the standard attention algorithm (Vaswani et al., 2017) achieves the highest accuracy but incurs the highest time and memory complexity. Second, StreamingLLM or Sink (Xiao et al., 2024b) retains only the KV cache of the initial and final tokens, resulting in low time and memory complexity, but this extreme approach leads to low accuracy on reasoning tasks (and other tasks (Tang et al., 2024)). Third, H2O (Zhang et al., 2023) theoretically offers low time and memory complexity, but its inability to utilize efficient attention kernels and the lack of page-level KV management makes it impractical, resulting in both low accuracy and infeasibility. Fourth, Quest achieves high accuracy and low time complexity but conservatively retains all KV cache, thus $O(N)$ memory complexity.

3 Algorithm Design

When directly applied to reasoning tasks, existing algorithms struggle with the "impossible trinity" of accuracy, time, and memory. For example, although the state-of-the-art Quest (Tang et al., 2024) achieves promising accuracy (Figure 6) with $O(L)$ time complexity, it requires storing the entire KV cache, thus $O(N)$ memory complexity (Figure 7). To break the "impossible trinity," we analyze the decoding stage of reasoning tasks 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 complexity, with accuracy comparable to Quest.

3.1 Reasoning Attention Pattern

By analyzing the attention map of the decoding stage of reasoning tasks, we discover two key char-

[2] https://www.reddit.com/r/OpenAI/comments/1frdwqk/your_longest_thinking_time_gpt4_o1_o1mini/

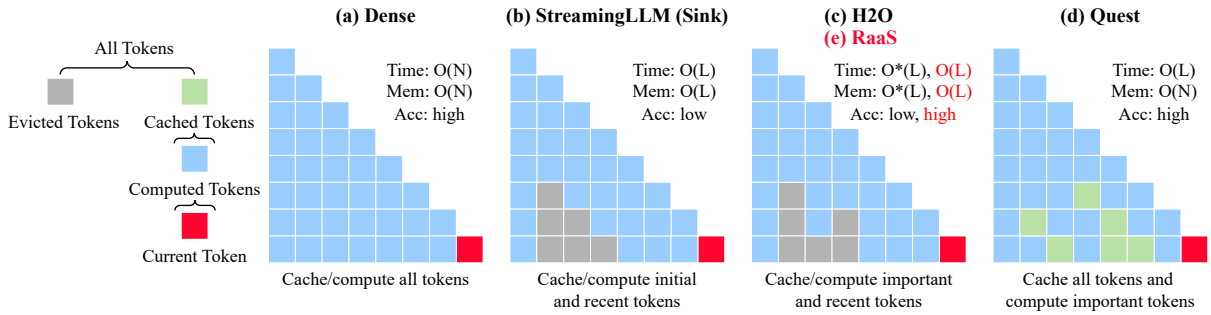


Figure 2: Comparison of sparsity-based algorithms. N indicates the sequence length while L indicates the cache budget where $L \ll N$. H2O's time and memory complexity are theoretical, as indicated by the asterisks. *RaaS* addresses the "impossible trinity" by achieving $O(L)$ complexity for both time and memory, with accuracy comparable to Dense on reasoning tasks.

acteristics (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 — similar to a water column in a waterfall (Figure 3 (a)) — must be carefully managed to prevent significant accuracy loss (Figure 6). 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 prefill prompts, such as user queries (Figure 3 (b)). Quest (Tang et al., 2024) retains the entire KV cache to avoid losing phoenix tokens, thus the $O(N)$ memory complexity.

We offer a possible explanation for the waterfall pattern or milestone tokens 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 token. 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[3] in Figure 4. First, tokens ①②③ serve as initial lemmas, which are crucial for subsequent deductions, corresponding to ①②③ columns in Figure 3. 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) only attend to tokens ④⑤.

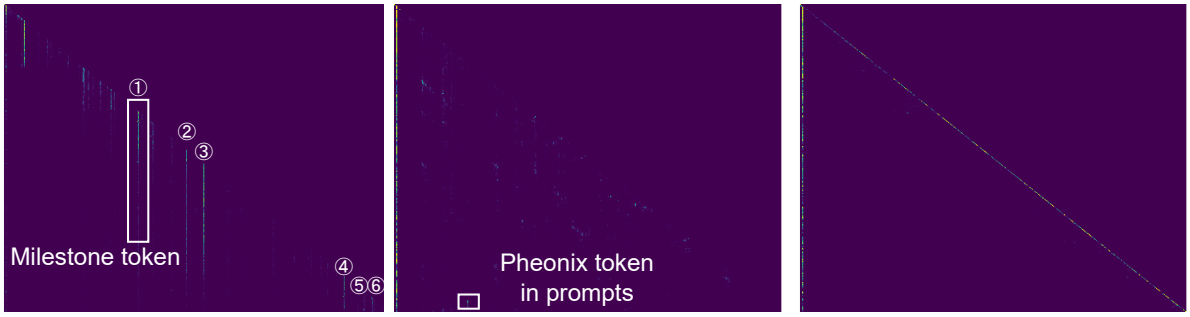
[3]Examples abound during the investigation of reasoning tasks, not limited to this one, and not limited to those extra examples in the appendix.

3.2 Design of *RaaS*

Based on the preceding observations, we propose *RaaS*, a simple yet effective algorithm that addresses the "impossible trinity" and consists of two main ideas. First, we identify and retain milestone tokens until they are no longer needed, using timestamps to track their importance. When a token receives an attention score above α (e.g., $\alpha = 0.01$), we assign it the latest timestamp. Milestone tokens always receive the latest timestamp until it becomes unimportant. When the cache is full, we evict tokens with the oldest timestamp. Second, we retain the KV cache 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 ensures that critical information is not lost during the decoding process.

To illustrate *RaaS* step by step, consider Figure 5. In the **first five steps**, the cache size limit is not reached (we can store at most 5 tokens). In the 4-th row, the second token is cached and computed, but its timestamp is not updated because its attention score is below α, which we consider insufficient for influencing the final result. However, in the 5th row, the second token's attention score exceeds α, and it is assigned the latest timestamp, 5. In the **last three steps**, the cache is full. In the 6th row, we evict the third token (the third column becomes gray) since it has the oldest timestamp. We then compute the remaining tokens and update their timestamps. A similar process is followed in the seventh and eighth rows.

The Choice of α. The choice α affects the distribution of tokens' timestamps. If α is small, too many tokens receive the latest timestamp, preventing effective differentiation of milestone tokens.



(a) Waterfall pattern (20%-25%) (b) Pheonix pattern (1%-2%) (c) StreamingLLM pattern (> 70%)

Figure 3: A "waterfall pattern," emerges in reasoning tasks. We manually inspect attention maps across 28 layers and 28 heads of Qwen2.5-Math-7B (Yang et al., 2024) on 100 MATH500 (Hendrycks et al., 2021) data points. We find (a) 20% to 25% maps with milestone tokens, (b) 1% to 2% maps with phoenix tokens that remain inactive for more than 128 decoding steps before becoming active again, (c) more than 70% "lazy" (Zhang et al., 2022) maps with StreamingLLM pattern. We used our best effort to balance the clarity and completeness of attention maps.

To convert the point \((0,3)\) from rectangular coordinates to polar coordinates, we need to find the values of \(r\) and \(\theta\). The formulas for converting from rectangular coordinates \((x,y)\) to polar coordinates \((r,\theta)\) are:

\[r = \sqrt{x^2 + y^2} \] ①
\[\theta = \tan^{-1}\left(\frac{y}{x}\right) \] ②

Given the point \((0,3)\), we have \(x = 0\) and \(y = 3\). Let's calculate \(r\) first: ③
\[r = \sqrt{0^2 + 3^2} = \sqrt{9} = 3 \]

Next, we need to find \(\theta\). The formula \(\theta = \tan^{-1}\left(\frac{y}{x}\right)\) is not directly useful here because it involves division by zero, which is undefined. Instead, we need to consider the position of the point \((0,3)\) in the coordinate plane. The point \((0,3)\) lies on the positive \(y\)-axis. Therefore, the angle \(\theta\) is \(\frac{\pi}{2}\).

So, the polar coordinates of the point \((0,3)\) are: ④
\[(r, \theta) = \left(3, \frac{\pi}{2}\right) \] ⑤

Thus, the final answer is: ⑥
\[\boxed{\left(3, \frac{\pi}{2}\right)} \]

Figure 4: We input the prefill tokens, "...Convert the point $(0, 3)$ to polar coordinates...", to Qwen 2.5 Math 7B Instruct and obtain the corresponding decode tokens in the figure. The red tokens represent the milestone tokens or "water columns" in the attention map.

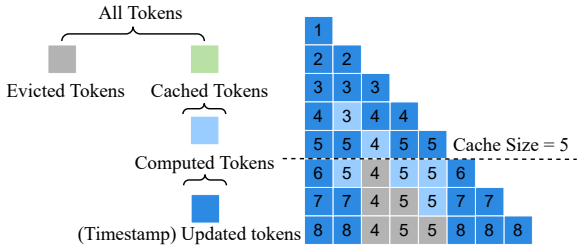


Figure 5: Illustration of *RaaS* by using an example that decodes eight tokens.

Conversely, if α is large, 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 $r = 50\%$ tokens with the highest attention scores in each decoding step, where $\alpha \approx 0.0001$ (The parameters α and r are two sides of the same coin) and This method provides effective results (Figure 9). Due to page limitations, we leave further exploration of the optimal α/r and its theoretical justification to future work.

3.3 Page-Based *RaaS*

Directly applying the version of *RaaS* in Section 3.2 presents two challenges. First, managing KV caches at the token level is inefficient, as small gaps in the cache complicate memory management and hinder GPU computation. Second, *RaaS* requires the attention scores of all tokens to update timestamps, but retrieving these scores is incompatible with optimized attention kernels like 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*[4]. First, we introduce a page-based caching system with a fixed page size of $page\_size = 16$. The timestamp management, as well as cache retention and eviction, is handled at the page level as in most of the modern inference engine (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 this attention score, we update the timestamp for each page and make eviction decisions at the page level. Various methods exist for selecting a representative K, such as those used in Quest (Tang

[4]From now on, whenever we use *RaaS* we refer to page-based *RaaS*.

et al., 2024) and ArkVale (Chen et al.). For fairness, we adopt the same representative selection method as in Quest.

4 Evaluation

4.1 Experiment Setup

We implement *RaaS* based on Hugging Face (Hugging Face) and Quest (Tang et al., 2024) with 2K lines of code. We port Quest from their public repository[5]. Next, we discuss the datasets, models, metrics, and the environment in which we carry out experiments.

Dataset. We take the first 200 questions from each of the following three open-sourced datasets for our benchmarks: GMS8k (Cobbe et al., 2021), MATH500 (Hendrycks et al., 2021), and AIME (AIME), to test the reasoning ability of language models. First, *GMS8k* (Cobbe et al., 2021) contains 8.5k high-quality, linguistically diverse grade school math word 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 mathematics 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 AIME (American Invitational Mathematics Examination) competition from 1983 to 2024, designed to challenge the most exceptional high school mathematics students in the United States. These problems cover various fields, such as algebra, geometry, and number theory.

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 (a 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 answer. For each data point, it is either correct or incorrect, and the overall accuracy is reported as the percentage of correctly solved problems across the entire dataset.

Models. We evaluate our algorithm using four popular models: Marco o1 (Zhao et al., 2024), Qwen2.5 Math 7B (Wang et al., 2024), Mistral Math 7B (Wang et al., 2024), and DeepScaleR 1.5B[6]. They are four of the most powerful open-sourced LLMs with long-reasoning capabilities.

Baselines. We compare *RaaS*'s 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 repo 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 hyper-threading and 1TB DRAM. We use Ubuntu 20.04 with Linux kernel 5.16.7 and CUDA 12.6. Unless stated otherwise, we set $\alpha = 0.0001$ 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 6). First, H2O and StreamingLLM exhibit poor accuracy under fixed cache budgets compared to others. StreamingLLM 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 all KVs while *RaaS* optimizes memory usage by specifically handling milestone tokens and using only $O(L)$ memory (Figure 7). 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, which negatively impacts 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 latency and memory consumption, yield-

[5] https://github.com/mit-han-lab/Quest. Accessed on Oct 2024.

[6] https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303013a4e2

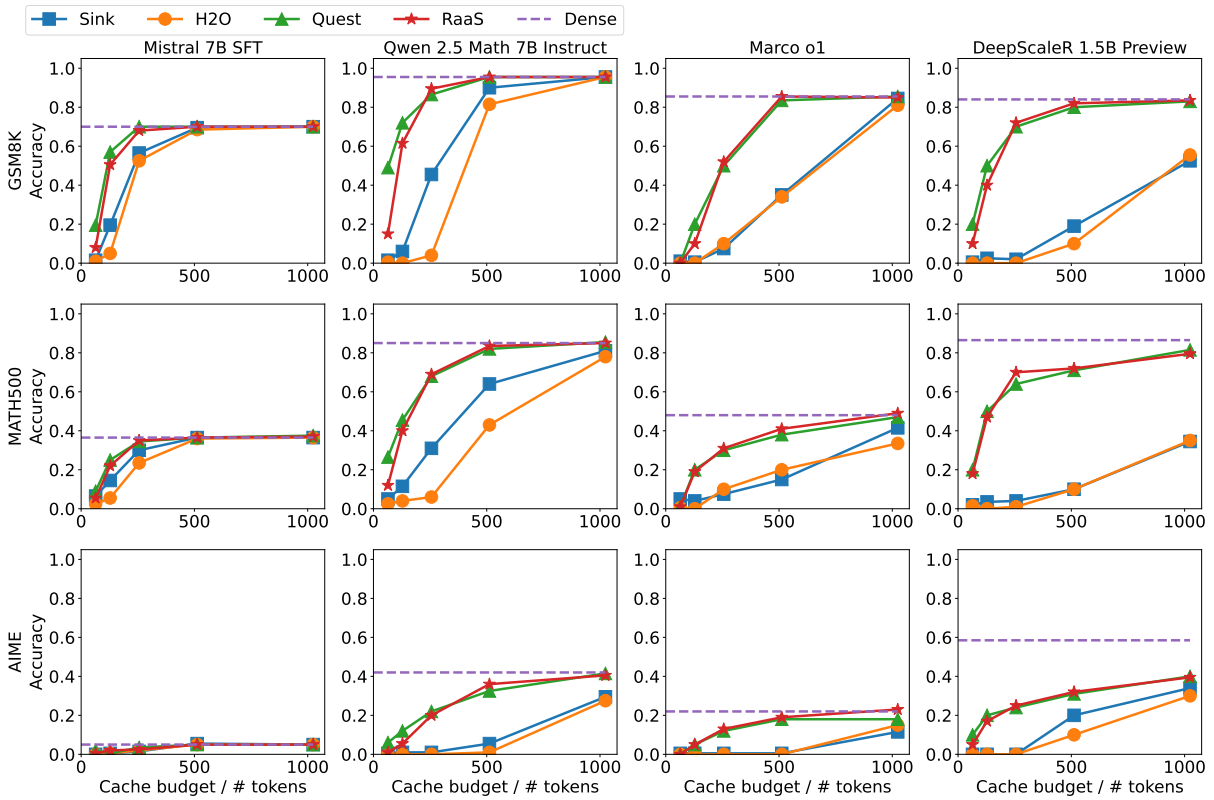


Figure 6: Accuracy vs. cache budget for five algorithms (legends) across three datasets (rows) and four models (columns). The y-axis shows the proportion of correctly solved problems among 200 data points, while the x-axis represents varying cache budgets: 64, 128, 256, 512, and 1024.

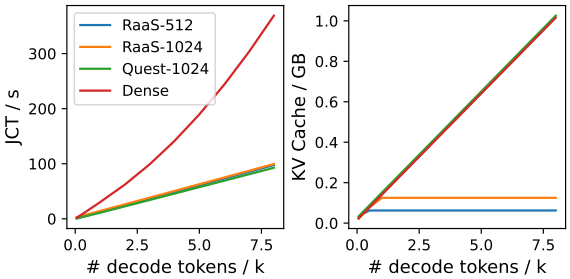


Figure 7: Latency and memory consumption of five algorithms running on Mistral Math 7B, using workloads with a fixed prefill length (128 tokens) and varying decode length (from 0 to 8k tokens).

ing two key observations from the experimental results (Figure 7). First, as the number of decode tokens increases, Dense's latency grows quadratically, while both *RaaS* and Quest exhibit linear latency growth. This is because Dense has $O(N^2)$ computation time, whereas *RaaS* and Quest have $O(NL)$ computation time, reducing each decoding step 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. In summary, while Dense and Quest have $O(N)$ memory complexity, *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 8 shows that discarding milestone tokens, as in H2O-128 and Sink-128, increases the decoding lengths. Analysis of the outputs reveals that while the model initially reasons correctly for the first few tokens (e.g., green tokens in Figure 8), 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.

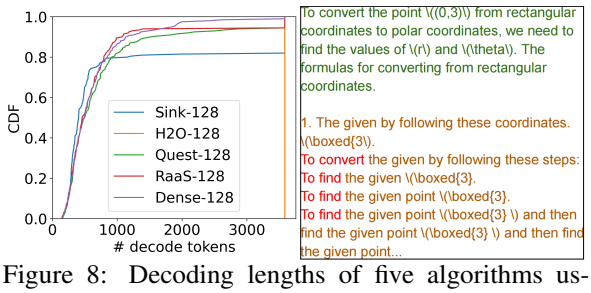


Figure 8: Decoding lengths of five algorithms using Qwen 2.5 Math 7B with 4k context length, on MATH500. Using H2O-128 (cache budget of 128 tokens) always reaches the 4k limit. On the right, we show a decoding example of H2O-128.

The Impact of α. The choice of α affects the distribution of tokens' timestamps, with $\alpha = 0.0001$

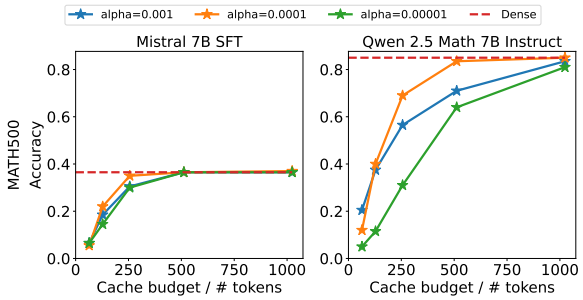


Figure 9: Accuracy of *RaaS* with different cache budgets and αs.

generally yielding optimal results, as shown in Figure 9. First, when α is small, too many tokens are assigned the latest timestamp, preventing effective differentiation of milestone tokens. Second, when α is large, most tokens are deemed irrelevant, potentially leading to the loss of milestone tokens.

5 Related Work

Several approaches have been proposed to reduce the computation and memory footprint of the KV cache during long-context inference. These can be categorized into two types: one that requires modifying 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, Muti-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 efforts discard the Transformer architecture entirely in favor of alternative models. Approaches such as Linear Attention and RNN-based models, including RWKV (Peng et al., 2023), RetNet (Sun et al., 2023), and Mamba (Gu and Dao, 2023), offer lower computational and memory costs. However, these models typically underperform compared to Transformer-based models in long-context scenarios.

5.2 KV Compression

The change of model architecture often requires significant pretraining or fine-tuning, whereas plug-and-play approaches, such as KV compression, are typically preferred in major application scenarios. Two primary types of KV compression have emerged: KV quantization and KV pruning.

KV Quantization. KV quantization approaches (Xiao et al., 2023; Yao et al., 2022; Dettmers et al., 2022) map higher precision KVs into lower ones, trading accuracy for savings in computation and memory. Recent studies have shown that due to the distinct element distributions in the KV cache, key and value caches require different quantization strategies to optimize performance on complex tasks (Zirui Liu et al., 2023).

KV Pruning. 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., 2024a), which posits that only around 5% of tokens are crucial during LLM inference. Thus, evicting less important tokens from the KV cache is a key strategy for reducing memory usage and accelerating inference. For example, StreamingLLM (Xiao et al., 2024b) 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) keeps the recent tokens and the top-k important tokens based on the attention score calculated within a local window. More recent works, such as Quest (Tang et al., 2024) and ARKVALE (Chen et al.), manage the KV cache at the page level, selecting the top-k important pages during each generation step to reduce computational time.

Our work presents a new trial of applying KV pruning in reasoning tasks, where the inference pattern is characterized by a short prefill and a long decode with a waterfall attention pattern. For the first time, it achieves true $O(L)$ time and memory complexity with high accuracy.

6 Conclusion

In this paper, we have identified a new waterfall attention pattern observed in the decode stage of reasoning tasks. Leveraging this pattern, we have proposed a sparsity-based algorithm named *RaaS* that achieves high accuracy while maintaining $O(L)$ time and $O(L)$ memory complexity. 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*'s 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.

Limited applicability of *RaaS*. *RaaS* is specifically designed for traditional reasoning tasks where a question is short (e.g., a mathematical query) but its answer is lengthy (e.g., a chain of reasoning followed by a final answer). The waterfall attention pattern primarily occurs during the decode stage, and phoenix tokens are frequently found in the question (prefill tokens). Thus, *RaaS* may lose crucial information when applied in other scenarios where the number of prefill tokens exceeds a certain threshold. In this case, we recommend using the combination of Quest (on prefill tokens) and RaaS (on decode tokens).

Evaluation on a limited set of models. Our evaluation is based on only four models. The presented results may be specific to these models and may not generalize to others. Although there are models with larger context lengths, such as Qwen2.5-Max and DeepSeek-r1, conducting a comprehensive evaluation across all such models is time- and resource-intensive. As noted by previous work (Zhong et al., 2024), if decoding a single token takes around 30 ms, decoding 16k tokens (a relatively small request) requires approximately 8 minutes on a single A100-80GB GPU. Evaluating 200 data points would take more than one day, being infeasible with only one available A100 GPU. Nonetheless, we believe that the core idea—the waterfall pattern and its underlying rationale— remains broadly applicable, and we plan to conduct additional experiments in the future.

Evaluation on limited inference lengths. Due to resource and time constraints, we are unable to conduct experiments with extremely long inference lengths, as this experimental setup would require months to complete end-to-end evaluations. However, through small-scale experiments, we observe that the waterfall attention pattern is universal across varying inference lengths.

Lack of exploration of representative selection for pages. For a fair comparison with Quest, we adopt the same representative selection algorithm used in Quest. However, given the distinct objectives of *RaaS*, there may be opportunities to design a more tailored representative selection algorithm that better handles false positives and negatives, potentially improving *RaaS*'s accuracy. Nonetheless, more sophisticated representative selection algorithms may introduce additional computational overhead. Therefore, we have not explored this aspect in our work described in this paper but plan to investigate the impact of different representative selection algorithms in future work.

References

AIME. 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, EMNLP 2023, Singapore, December 6-10, 2023*, pages 4895–4901. Association for Computational Linguistics.

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, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 3119–3137. Association for Computational Linguistics.

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*, abs/2406.02069.

Renze Chen, Zhuofeng Wang, Beiquan Cao, Tong Wu, Size Zheng, Xiuhong Li, Xuechao Wei, Shengen Yan, Meng Li, and Yun Liang. Arkvale: Efficient generative llm inference with recallable key-value eviction. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

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*, abs/2110.14168.

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 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 1280–1297. Association for Computational Linguistics.

Tri Dao. 2024. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Gpt3.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

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*, abs/2407.14057.

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*, abs/2312.10997.

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 *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *CoRR*, abs/2312.00752.

Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2024. Lm-infinite: Zero-shot extreme length generalization for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 3991–4008. Association for Computational Linguistics.

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 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

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*, abs/2406.17565.

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 (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 7036–7050. Association for Computational Linguistics.

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 *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

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*, abs/2404.12457.

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, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pages 611–626. ACM.

Huayang Li, Yixuan Su, Deng Cai, Yan Wang, and Lemao Liu. 2022. A survey on retrieval-augmented text generation. *CoRR*, abs/2202.01110.

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 *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

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 *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

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. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

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 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 4089–4100. Association for Computational Linguistics.

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, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 18724–18741. Association for Computational Linguistics.

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, Bartlomiej 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 2023, Singapore, December 6-10, 2023*, pages 14048–14077. Association for Computational Linguistics.

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, MLSys 2023, Miami, FL, USA, June 4-8, 2023*. mlsys.org.

Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. 2023. In-context retrieval-augmented language models. *Trans. Assoc. Comput. Linguistics*, 11:1316–1331.

Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *CoRR*, abs/1911.02150.

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*, abs/2307.08621.

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 *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

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 *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 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*, abs/2410.09671.

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 *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

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 *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 38087–38099. PMLR.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. 2024a. Duoattention: Efficient long-context LLM inference with retrieval and streaming heads. *CoRR*, abs/2410.10819.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024b. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

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*, abs/2409.12122.

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 *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Xuan Zhang, Fengzhuo Zhang, Cunxiao Du, Chao Du, Tianyu Pang, Wei Gao, and Min Lin. 2022. LightTransfer: your long-context llm is secretly a hybrid model with effortless adaptation. *CoRR*, abs/2410.13846.

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 *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

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*, abs/2411.14405.

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 *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

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 *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, pages 193–210. USENIX Association.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2023. Kivi : Plug-and-play 2bit kv cache quantization with streaming asymmetric quantization.

A More Examples for Milestone Tokens

This section presents more examples of milestone tokens in Figure 10 and Figure 11. For all examples, we input the prefill tokens to Qwen 2.5 Math 7B Instruct and obtain the corresponding decode tokens, as shown in the figure. The tokens marked red represent the milestone tokens or "water columns" in the attention map. Although we only show a few examples here, the milestone tokens or waterfall patterns abound in reasoning tasks.

Prompt: Define
\[p = \sum_{k = 1}^\infty \frac{1}{k^2} \quad \text{and} \quad q = \sum_{k = 1}^\infty \frac{1}{k^3}.\]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:
\[\sum_{n=2}^\infty \frac{1}{n^2} - \sum_{n=2}^\infty \frac{1}{n^3}. \]
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:
\[(p - 1) - (q - 1) = p - q. \] ④
Thus, the double sum \(\sum_{j=1}^\infty \sum_{k=1}^\infty \frac{1}{(j+k)^3}\) in terms of \(p\) and \(q\) is: ⑤
\[\boxed{p - q}. \]

Figure 10: Milestone example.

B System Implementation

We implement *RaaS* based on Hugging Face (Hugging Face) and Quest (Tang et al., 2024). Our implementation extends the standard transformer architectures (Dense) by incorporating the following additional components.

First, for each page in the KV cache, a representative key is selected to compute the estimated attention score between the page and the query. To ensure consistency in evaluation, our page-based *RaaS* adopts the same representative selection algorithm as that used in Quest.

Second, a timestamp is associated with each KV page to record the last *time* at which the page received an estimated attention score exceeding the threshold α (e.g., $\alpha = 0.01$). The term *time* here refers to any monotonically increasing attribute. In our implementation, we use the number of tokens (including both prefill and decode tokens) as this attribute.

Third, when the KV cache reaches its maximum capacity, an eviction algorithm evicts the KV page with the earliest recorded *time*. Pages corresponding to prefill tokens are exempt from eviction.

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:
\[2 = \frac{6}{3} \quad \text{and} \quad 1 = \frac{3}{3} \]

Now we can add the fractions: ⑥
\[\frac{6}{3} + \frac{5}{3} + \frac{3}{3} = \frac{6 + 5 + 3}{3} = \frac{14}{3} \] ⑦

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

Figure 11: Milestone example.

The selection of representative keys and the computation of estimated attention scores are interleaved with the self-attention computation in each layer, aligning with Quest's implementation. The updates of timestamps and the eviction of KV pages are performed after each autoregressive iteration, with all layers processed in a batched manner, resulting in negligible time overhead, as shown in Figure 7.

C 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[7], 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.

[7] https://huggingface.co/