

# ZoomR: Memory Efficient Reasoning through Multi-Granularity Key Value Retrieval

David H. Yang<sup>1</sup>, Yuxuan Zhu<sup>1</sup>, Mohammad Mohammadi Amiri<sup>1</sup>,  
Keerthiram Murugesan<sup>2</sup>, Tejaswini Pedapati<sup>2</sup>, Subhajit Chaudhury<sup>2</sup>, Pin-Yu Chen<sup>2</sup>

<sup>1</sup>Rensselaer Polytechnic Institute <sup>2</sup>IBM Research  
{yangd13, zhuy27, mamiri}@rpi.edu tejaswinip@us.ibm.com  
{keerthiram.murugesan, subhajit, pin-yu.chen}@ibm.com

## Abstract

Large language models (LLMs) have shown great performance on complex reasoning tasks but often require generating long intermediate thoughts before reaching a final answer. During generation, LLMs rely on a key-value (KV) cache for autoregressive decoding. However, the memory footprint of the KV cache grows with output length. Prior work on KV cache optimization mostly focus on compressing the long input context, while retaining the full KV cache for decoding. For tasks requiring long output generation, this leads to increased computational and memory costs. In this paper, we introduce ZoomR, a novel approach that enables LLMs to adaptively compress verbose reasoning thoughts into summaries and uses a dynamic KV cache selection policy that leverages these summaries while also strategically “zooming in” on fine-grained details. By using summary keys as a coarse-grained index during decoding, ZoomR uses the query to retrieve details for only the most important thoughts. This hierarchical strategy significantly reduces memory usage by avoiding full-cache attention at each step. Experiments across math and reasoning tasks show that our approach achieves competitive performance compared to baselines, while reducing inference memory requirements by more than 4×. These results demonstrate that a multi-granularity KV selection enables more memory efficient decoding, especially for long output generation.

## 1 Introduction

Large language models (LLMs) have demonstrated remarkable proficiency in complex reasoning tasks such as mathematics and coding (OpenAI et al., 2024a; DeepSeek-AI et al., 2025). This success can be attributed to their ability to generate long chain of thoughts before arriving at a final solution. However, this verbosity introduces a significant computational and memory bottleneck. The

autoregressive decoding process relies on a key-value (KV) cache that stores hidden states from previously generated tokens to enable efficient self-attention, and this cache grows linearly with the sequence length. This presents two critical challenges for long-output generation. First, the KV cache consumes a large amount of GPU memory, and second, the computational cost of attending to the ever-growing cache makes each subsequent token progressively slower to generate. For instance, generating a 16K-token response with a Llama-3.1-8B (Grattafiori et al., 2024) model for a batch size of 8 requires over 16GB of GPU memory for the KV cache, making deployment more challenging on consumer GPUs with less memory.

To address these challenges, several efficient reasoning strategies have been proposed (Sui et al., 2025). One line of work focuses on enabling the LLM to generate fewer tokens using techniques like prompt engineering (Han et al., 2025; Aytes et al., 2025), specialized training algorithms (Team et al., 2025; Xia et al., 2025), and latent space reasoning (Hao et al., 2024a; Shen et al., 2025a). For these works, the primary objective is to speed up inference, and may implicitly reduce GPU memory consumption. Another direction, which we follow in this paper, aims to optimize the KV cache itself. While several methods for cache eviction or compression exist, most are optimized for long input contexts during the prefill stage, when the model processes the prompt, (Li et al., 2024; Tang et al., 2024; Sun et al., 2025; Zhu et al., 2025a), rather than addressing the unique memory challenges of long-output generation during decoding. Existing dynamic KV cache methods lose critical long-range information in long-output generation, resulting in poor performance on complex reasoning tasks (Xiao et al., 2024; Zhang et al., 2023; Chen et al., 2025). This leaves a crucial gap for an efficient cache management strategy tailored specifically for lengthy, generated reasoning chains,

while maintaining high performance.

To address this gap, we propose ZoomR, a method inspired by established principles of human memory and attention. When solving complex problems, humans leverage hierarchical memory representations, maintaining schematic overviews of past information while selectively retrieving detailed specifics only when contextually relevant (Bartlett, 1932). ZoomR applies this strategy to LLMs by maintaining both compressed summaries (coarse-grained representations) of the generation history and the original detailed text, accessing fine-grained information only when the current context demands higher fidelity. We first fine-tune reasoning models to generate summaries of thoughts after each paragraph. During inference, ZoomR uses the current query and summary keys across all attention heads to identify the most important segments of the past thoughts. Then, a consensus is performed to retrieve the most important full-resolution details. By doing so, ZoomR only selects a small number of KVs to keep on GPU for attention computation, while offloading the full KV cache to CPU. Our implementation involves a one-time fine-tuning step to teach the model to summarize its own history, followed by a training-free dynamic cache selection policy at inference time.

Our contributions are three-fold. First, we introduce ZoomR, a dynamic, multi-granularity KV cache management technique designed to enable memory efficient long-output reasoning. Second, we present a practical framework that combines lightweight fine-tuning for summary generation with an efficient, approximate attention score-based retrieval mechanism for inference. Third, we empirically validate ZoomR on reasoning models, Qwen and Llama (Qwen et al., 2025; Grattafiori et al., 2024), across challenging reasoning benchmarks, AIME2025 and MATH500. Our results demonstrate that ZoomR achieves performance comparable to a standard full KV cache while saving more than 4× the GPU memory usage during inference.

## 2 Background

We investigate the efficiency of autoregressive inference for LLMs in long-generation reasoning tasks. In this setting, the model is given a prompt  $\mathbf{X}$  of length  $N_p$  and is tasked with generating a lengthy response  $\mathbf{Y}$  of length  $N_g$ ,  $N_g \gg N_p$ . Modern reasoning models trained using chain-of-thought

prompting significantly improve performance over standard instruction-tuned models. However, this improvement comes at the cost of generating much longer sequences. The output  $\mathbf{Y}$  is often structured into a detailed reasoning or “thought” section, followed by a final “solution” section, where the length of the thought process is typically much greater than that of the final answer. With recent advancements, models are capable of generating outputs that extend to tens of thousands of tokens.

At the core of the Transformer architecture is the self-attention mechanism, which is central to the decoding process. During autoregressive generation, a token is produced at each timestep  $t$ . The model has  $N_L$  layers and  $H$  attention heads. Let  $\mathbf{x}_t \in \mathbb{R}^{d_{\text{emb}}}$  be the input embedding for the token. For a given position  $t$ , layer  $l$ , and head  $h$ , the input  $\mathbf{x}_t$  is projected into query, key, and value vectors  $\mathbf{q}_t$ ,  $\mathbf{k}_t$ , and  $\mathbf{v}_t$ , of dimension  $d$ :

$$\mathbf{q}_t = \mathbf{x}_t \mathbf{W}_q, \quad \mathbf{k}_t = \mathbf{x}_t \mathbf{W}_k, \quad \mathbf{v}_t = \mathbf{x}_t \mathbf{W}_v,$$

where  $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{d_{\text{emb}} \times d}$  are learned projection matrices. For ease of notation, we have dropped the index for layers and heads. To maintain context from all previous tokens, both from the prompt and the ongoing generation, the keys and values from all past steps are stored in a KV cache. At each step  $t$ , the newly computed key  $\mathbf{k}_t$  and value  $\mathbf{v}_t$  are appended to the cached matrices  $\mathbf{K}_{t-1}$  and  $\mathbf{V}_{t-1}$ :

$$\mathbf{K}_t = [\mathbf{K}_{t-1}; \mathbf{k}_t], \quad \mathbf{V}_t = [\mathbf{V}_{t-1}; \mathbf{v}_t],$$

where  $\mathbf{K}_t, \mathbf{V}_t \in \mathbb{R}^{(N_p+t) \times d}$ . The attention output  $\mathbf{A}$  is then computed as the scaled dot product between the query and cached keys, followed, by a softmax operation to obtain attention weights, and using them to aggregate the corresponding values:

$$\mathbf{A}(\mathbf{q}_t, \mathbf{K}_t, \mathbf{V}_t) = \text{softmax}\left(\frac{\mathbf{q}_t \mathbf{K}_t^\top}{\sqrt{d}}\right) \mathbf{V}_t \in \mathbb{R}^d.$$

This standard decoding process presents two major efficiency challenges, particularly as the generation length grows. First, the computational cost of generating each new token scales with the sequence length. The matrix-vector multiplication  $\mathbf{q}_t \mathbf{K}_t^\top$  scales linearly with the length of the KV cache, resulting in quadratic complexity with respect to the total generation length. As the generation proceeds and the cache grows, each subsequent token becomes progressively more expensive

to compute, making the overall inference process slow for long outputs. The second challenge is in GPU memory requirements. The primary memory bottleneck during LLM inference is the KV cache. The size of the cache matrices,  $\mathbf{K}_t$  and  $\mathbf{V}_t$ , grows linearly with the sequence length. For a model with multiple layers and attention heads, this memory footprint can become prohibitively large, limiting the maximum sequence length that can be processed on available hardware.

### 3 Methodology

In this section, we present ZoomR, a method that enables memory-efficient reasoning through multi-granularity KV retrieval. We begin by motivating our approach followed by its two main components: (1) fine-tuning an LLM to learn how to generate summaries on the fly, and (2) a dynamic KV cache selection policy for memory efficient decoding.

#### 3.1 Motivation

As LLM generation length increases, the KV cache creates practical limitations for extended reasoning tasks. Existing approaches attempt to address this challenge through various strategies, but often introduce significant limitations that compromise reasoning quality. Streaming-based approaches like StreamingLLM (Xiao et al., 2024), retain attention sinks and a sliding window of recent KV cache entries, but suffer from the “lost-in-the-middle” problem, where critical intermediate reasoning steps are permanently discarded. Dynamic token selection methods like H2O (Zhang et al., 2023) preserve important tokens through attention-based scoring mechanisms. However, these approaches face two critical limitations. First, computing token-level importance scores at each generation step incurs substantial computational overhead. Second, token-level selection operates at a granularity that may miss semantic coherence, potentially fragmenting coherent reasoning segments.

Recent work has also explored using compressed summaries to reduce context size while preserving semantic information (Zhang et al., 2025; Yan et al., 2025). While summarization provides meaningful compression, restricting attention exclusively to summary representations creates an information bottleneck. The key insight driving our approach is that different reasoning steps require access to historical context at different levels of granularity (see Figure 1). When tackling a challenging mathemat-

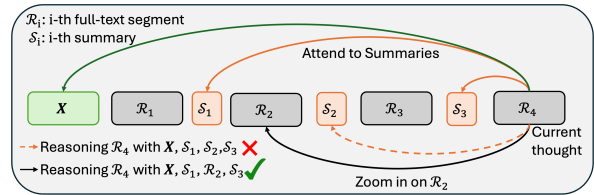


Figure 1: Fine-grained details are often needed in order to generate the correct reasoning. Attending to only a compressed source of information like summaries can lead to incorrect intermediate steps.

cal problem, the solution can draw on multiple lemmas and theorems developed throughout the reasoning process. Since summarization is inherently a lossy compression strategy, there may be steps during generation where high-level summaries fall short. For certain reasoning steps, accessing the full details becomes essential, whether to verify a subtle constraint, apply a specific technique, or build upon a particular intermediate result.

#### 3.2 Reasoning with Summarization

By default, LLMs do not explicitly summarize their reasoning after each paragraph. To induce this behavior in smaller models, we fine-tune them to generate summaries after each reasoning segment. To achieve this, we first augment the Bespoke-17K reasoning dataset (Labs, 2025) with paragraph-level summaries.

Specifically, for each example, we segment the text within the thought token boundaries into paragraphs. Then, we use a larger model, Llama3-70B to summarize each segment into a concise summary. To create distinct, modular summaries and maintain computational efficiency during data preparation, each summary is generated based only on the prompt instruction and its corresponding paragraph. Summary delimiter tokens  $\langle \text{begin\_of\_summary} \rangle$  and  $\langle \text{end\_of\_summary} \rangle$  are inserted before a summary begins and after it ends, respectively, to explicitly mark the boundaries of each summary ((see Appendix A.1 for an example). We then fine-tune a base reasoning model with LoRA (Hu et al., 2021) to enable summary generation during inference. While our approach requires an initial training stage, we note that larger models like Llama3-70B, GPT4, etc. (Grattafiori et al., 2024; OpenAI et al., 2024b), if prompted, can generate summaries in a specific format, and the additional fine-tuning step may be unnecessary. Thus, the historical context is partitioned into corresponding full-text and

summary segments.

### 3.3 The ZoomR Algorithm

Building upon the model’s ability to generate summaries, we introduce the core algorithm of ZoomR. The objective is to dynamically construct a multi-granularity KV cache at each reasoning step, focusing on the most important parts of the generation context and retaining a compressed view of the rest. This process unfolds in four stages: representation, scoring, aggregation, and context construction (see Figure 2).

The output of the fine-tuned model partitions the generation context into two types of segments. The regular segments ( $\mathcal{R}_i$ ) are sets of token indices corresponding to the original, detailed segments of the reasoning history and each summary segment ( $\mathcal{S}_i$ ) is a set of token indices for the compressed summary of a regular segment, where  $i$  denotes the summary number and its corresponding regular segment. The ZoomR algorithm operates by dynamically selecting a mix of these two segment types.

#### Representing Context with Mean Summary Keys.

To efficiently score the importance of historical segments, each summary segment  $\mathcal{S}_i$  is first compressed into a single representative vector for each attention head in each layer. This vector, termed the mean summary key, is computed by averaging the key vectors of all tokens within the summary. This is motivated by the observation that keys within a short, coherent text segment exhibit locality (Sun et al., 2025). Formally, for each summary Segment  $\mathcal{S}_i$ , layer  $l \in \{1, \dots, N_L\}$ , and head  $h \in \{1, \dots, H\}$ , the mean summary key is:

$$\bar{\mathbf{k}}_i^{(l,h)} = \frac{1}{|\mathcal{S}_i|} \sum_{j \in \mathcal{S}_i} \mathbf{k}_j^{(l,h)},$$

where  $\mathbf{k}_j^{(l,h)}$  denotes the corresponding key vector for layer  $l$  and head  $h$ . These mean summary keys are cached on the GPU for rapid access during decoding.

**Per-Head Importance Scoring and Voting.** At a given generation step  $t$ , the model’s current query vector,  $\mathbf{q}_t^{(l,h)}$ , holds information about the context needed to generate the next token. Since query dynamics change during generation, the notion of importance and KV cache selection is greatly dependent on  $\mathbf{q}_t^{(l,h)}$ . We use this query to perform a per-head importance assessment. Specifically, for

each head, we compute an approximate attention score through an inner product between the query and every mean summary key:

$$\alpha_i^{(l,h)} = (\mathbf{q}_t^{(l,h)})^\top \bar{\mathbf{k}}_i^{(l,h)}.$$

Each attention head then “votes” for the summaries it deems most important by selecting the indices of the top- $k$  highest-scoring summaries,

$$\mathcal{I}_{\text{top-k}}^{(l,h)} = \arg \text{top-k}_{i \in \{1, \dots, N_t\}}(\alpha_i^{(l,h)}),$$

where  $N_t$  is the number of summaries generated up until time  $t$ . This initial filtering identifies a candidate set of important segments from the perspective of each head.

**Global Aggregation and Consensus.** To form a global decision, the individual votes from all heads and layers are aggregated. We first collect the union of all unique summary indices selected across heads and layers, thereby forming a global candidate set:

$$\mathcal{I}_{\text{all}} = \bigcup_{l=1}^{N_L} \bigcup_{h=1}^H \mathcal{I}_{\text{top-k}}^{(l,h)}.$$

Next, we count the total number of votes each summary in this candidate set received:

$$v_i = \sum_{l=1}^{N_L} \sum_{h=1}^H \mathbb{I}(i \in \mathcal{I}_{\text{top-k}}^{(l,h)}),$$

where  $\mathbb{I}(\cdot)$  is the indicator function. The  $c$  summaries with the highest vote counts are designated as the consensus set,  $\mathcal{I}_c$ . These represent segments identified as highly important by a majority of attention heads and layers.

#### Constructing the Final Multi-Granularity Context.

The final KV cache is constructed by combining context at different resolutions based on the consensus results. Specifically, for summaries in the consensus set  $\mathcal{I}_c$ , we “zoom in” by including their corresponding original, full-text segments ( $\mathcal{R}_i$ ). This ensure that segments consistently identified as important across heads and layers are restored to full resolution, preserving fine-grained information critical for accurate intermediate reasoning while maintaining compression elsewhere. For the remaining summaries in the global candidate set, defined as  $\mathcal{I}_s = \mathcal{I}_{\text{all}} \setminus \mathcal{I}_c$ , we retain their compressed summary representations,  $\mathcal{S}_i$ . These segments are deemed relevant by at least one head but lack model-wide consensus.

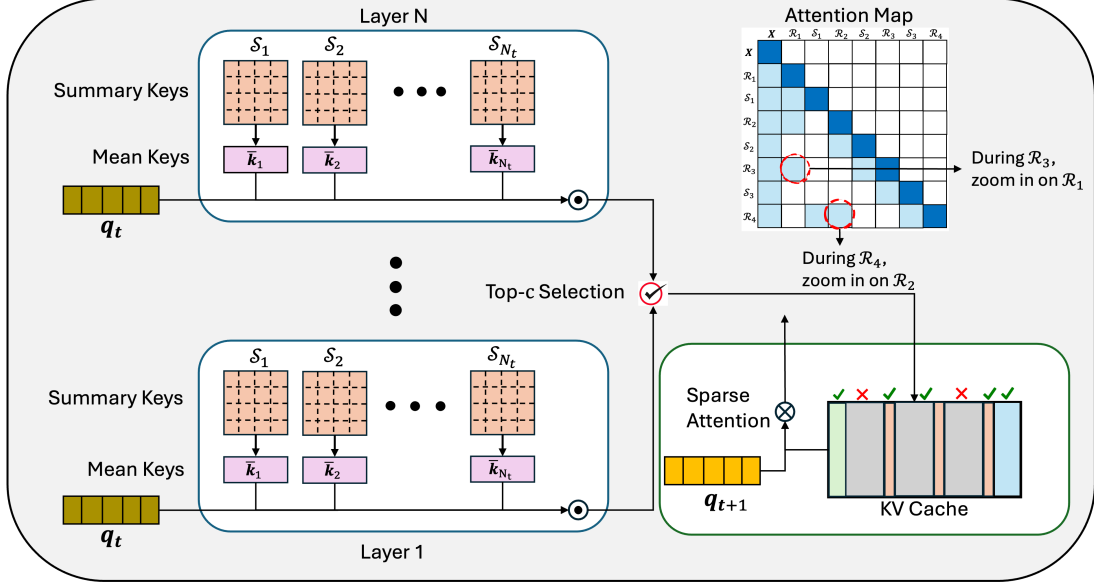


Figure 2: An overview of the ZoomR workflow. Mean summary keys are computed and cached on GPU. When the KV cache selection needs to be updated, approximate attention scores are calculated across all layers. A global top- $c$  selection identifies which summaries to zoom into. The final sparse KV cache loaded to GPU is a multi-granularity mix of an attention sink (green), selected detailed segments, summaries, and fixed recent window (blue). The symbolic attention map illustrates the dynamic nature of ZoomR during decoding.

To maintain fundamental coherence, this dynamically selected context is augmented with two critical, static components: the initial prompt tokens ( $\mathcal{I}_p$ ), which act as an attention sink, and a sliding window of the most recently generated tokens ( $\mathcal{I}_w$ ). The final set of token indices,  $\mathcal{I}_f$ , for the reduced KV cache is therefore obtained using the union of these components:

$$\mathcal{I}_f = \mathcal{I}_p \cup \mathcal{I}_w \cup \left( \bigcup_{i \in \mathcal{I}_c} \mathcal{R}_i \right) \cup \left( \bigcup_{i \in \mathcal{I}_s} \mathcal{S}_i \right).$$

At the subsequent generation step, the attention mechanism will compute over the keys and values corresponding only to the indices in  $\mathcal{I}_f$ . This dynamic, global selection process allows the model to form a holistic view of context relevance, enabling it to flexibly adjust its attentional focus between high-level summaries and fine-grained details as needed depending on the current query. The full algorithm for ZoomR can be found in Appendix A.2.

### 3.4 System Implementation and Inference

The practical application of ZoomR involves a memory management strategy that dynamically moves KV cache data between high-capacity CPU memory and fast GPU memory. The process is divided into a one-time prefill stage and a repeated decoding stage.

**Prefill Stage.** During the prefill stage, the input prompt  $X$  of length  $N_p$  is processed to generate the initial KV cache. To conserve GPU memory, we offload these initial KVs to CPU memory. Prompt tokens are crucial to maintain coherent LLM inference (Xiao et al., 2024), and this is accentuated in complex reasoning tasks. Thus, we choose to retain the full KV cache from the prefill stage. In many mathematical reasoning benchmarks,  $N_p \ll N_g$ , so the relative cost is minimal.

**Decoding Stage.** To optimize for memory efficiency, at each generation step, only the subset of the KV cache specified by the index set  $\mathcal{I}_f$  is loaded from CPU to GPU. After the attention computation for the current token is complete, the newly generated KV pair is appended to the full cache on the CPU. While this CPU-GPU data transfer introduces latency dependent on Peripheral Component Interconnect Express (PCIe) bandwidth, it enables reasoning over contexts far exceeding the available GPU memory.

To optimize performance, we introduce two key efficiency improvements. First, recomputing the selection set  $\mathcal{I}_f$  at every step is computationally expensive. We only update the selection set at semantic boundaries, such as end-of-sentence, amortizing the cost of selection over multiple decoding steps (see Appendix A.3 for details). Second, to

Table 1: Performance comparison of Llama and Qwen models across GPQA Diamond, AIME2025, and MATH500 benchmarks using various approaches. Results are reported in percent (%).

Model	Approach	GPQA Diamond	AIME2025	MATH500	Average
Llama	Vanilla	48.9	26.7	88.9	54.8
	StreamingLLM	34.7	13.3	74.7	40.9
	H2O	36.8	16.7	76.5	43.3
	SumR	37.3	16.7	77.8	43.9
	<b>ZoomR</b>	<b>43.4</b>	<b>26.7</b>	<b>83.0</b>	<b>51.0</b>
Qwen	Vanilla	49.0	30.0	92.5	57.2
	StreamingLLM	37.9	13.3	77.9	43.0
	H2O	39.1	16.7	79.4	45.1
	SumR	37.6	13.3	80.1	43.7
	<b>ZoomR</b>	<b>42.8</b>	<b>23.3</b>	<b>87.8</b>	<b>51.3</b>

minimize peak GPU memory usage and hide data transfer latency, we implement a pipelined, layer-by-layer execution strategy. For each layer of the model, the required KV cache slices are transferred to the GPU. Once the layer’s computation is finished, its KVs are asynchronously transferred back to the CPU while the KVs for the next layer are prefetched. This overlaps data transfer with computation, reducing GPU idle time.

### 3.5 Theoretical Memory Analysis

ZoomR’s efficiency comes from only loading a small subset of the KV cache from CPU to GPU at each decoding step. We quantify this by comparing the memory footprint of a full KV cache with that of ZoomR’s active sparse cache on the GPU.

The KV cache size selected by ZoomR corresponds to the number of tokens in the selection set  $\mathcal{I}_f$ . Let us consider each component in  $\mathcal{I}_f$ . Define the number of prompt tokens as  $N_p$ , the consensus selected  $c$  regular segments of average length  $\bar{L}_R$ ,  $(|\mathcal{I}_{\text{all}}| - c)$  summary segments of average length  $\bar{L}_S$ , and size of the recent window as  $N_w$ . In the worst case, the number of selected summaries  $|\mathcal{I}_{\text{all}}|$  is bounded by the total number of summaries generated during decoding. During generation, the full KV cache stores all previously generated tokens, reaching a size of  $N_g$  at the final step. Thus, the memory savings (MS) can be computed as:

$$\text{MS} = \frac{N_p + N_g}{N_p + c \cdot \bar{L}_R + (|\mathcal{I}_{\text{all}}| - c) \cdot \bar{L}_S + N_w}$$

For  $N_p = 512$ ,  $N_g = 16384$ ,  $N_w = 512$ ,  $c = 2$ ,  $\bar{L}_R = 250$ ,  $\bar{L}_S = 20$ , and  $|\mathcal{I}_{\text{all}}| = 80$ , the memory savings is  $5.48\times$ .

Crucially, the denominator grows sub-linearly with the sequence length as the number of sum-

maries increases at a smaller rate. This ensures that memory savings improve as the generation length increases. For instance, doubling  $N_g$  to 32K yields savings of approximately  $7.11\times$ .

## 4 Experiments

In this section we present the experiment setup and evaluate ZoomR using two reasoning models: Qwen2.5-7B and Llama-3.1-8B. For each model, we perform supervised fine-tuning with a max context length of 16K tokens using the augmented Bespoke-17K dataset. We refer to these fine-tuned models as the vanilla baselines. Specifically, we use the distilled reasoning models from R1-Distill, Deepseek-R1-Distill-Qwen2-7B and Deepseek-R1-Distill-Llama3-8b (DeepSeek-AI et al., 2025) as the base models for fine-tuning.

We compare ZoomR against three other baseline approaches: StreamingLLM, H2O, and SumR. StreamingLLM uses a static KV cache selection policy based on fixed attention sinks and a recent sliding window, while H2O uses a dynamic approach that selects tokens based on importance scores at each generation step. To ensure a fair comparison between ZoomR, StreamingLLM, and H2O, we first compute the GPU budget used by ZoomR for each task, and then set the equivalent budget for StreamingLLM and H2O. We also introduce SumR, a simplified variant of ZoomR that retains only the summary tokens and does not perform any detailed token retrieval. We evaluate all methods on both math and reasoning tasks, including MATH500, GPQA Diamond, and AIME2025. All experiments are run on NVIDIA H100 GPUs.

**Results.** As shown in Table 1, ZoomR significantly outperforms baseline approaches for all

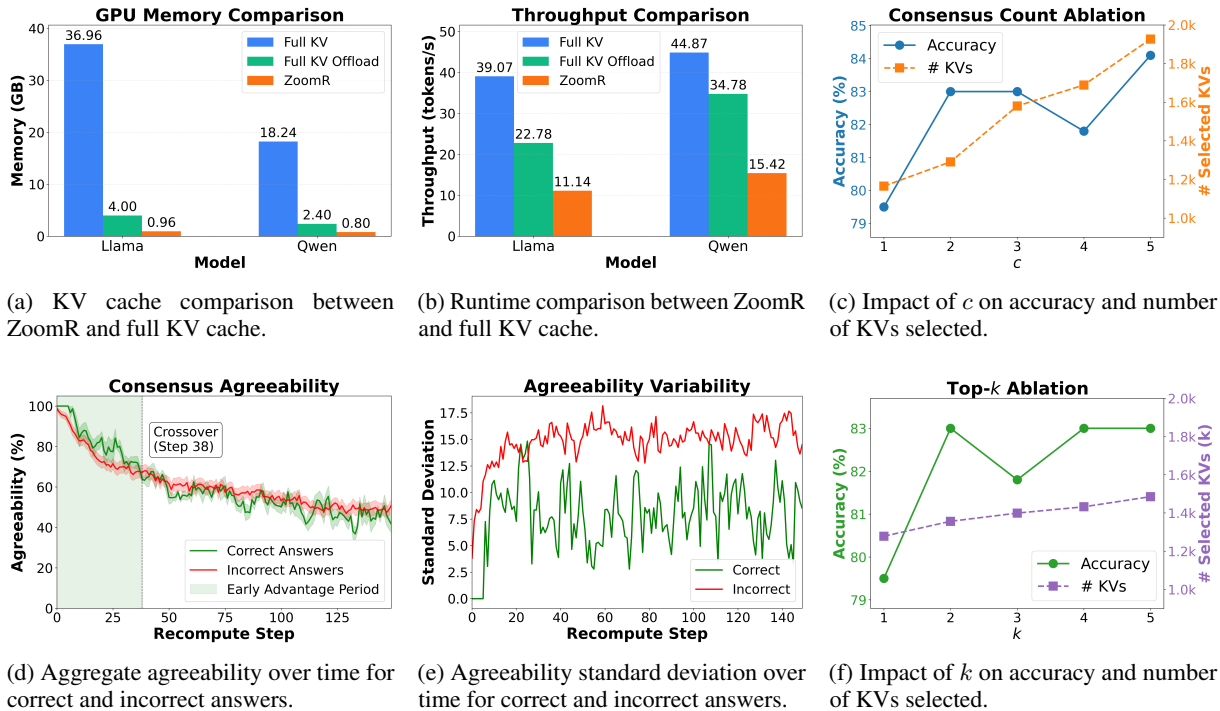


Figure 3: Efficiency, consensus, and ablation studies. Fig. (a,b) shows the GPU memory and throughput during inference, respectively. Fig. (d,e) show consensus dynamics between correct and incorrect answers over time. Fig. (c,f) illustrates the relationships between  $c$  and  $k$  on performance and KV cache selection size.

tasks, achieving accuracy close to the full KV cache. For the challenging AIME2025 benchmark, ZoomR is particularly effective, matching the performance of the full KV cache with the Llama model while remaining highly competitive using Qwen. Notably, these results demonstrate the benefits of dynamically zooming in to detailed context. ZoomR sees an 8% average accuracy improvement over SumR, indicating that only selecting summaries leads to loss of information, thereby impacting performance.

ZoomR is significantly more memory efficient than the vanilla baselines. We compare against both cases of full KV cache on GPU and offloaded to CPU. Figure 3a shows the GPU memory utilization during inference generation in these cases using a 16K generation length and batch size of 16. Note that we exclude the model weights’ contribution to memory. ZoomR saves more than  $20\times$  GPU memory for both Llama and Qwen based models compared to using the full KV cache stored on GPU, and more than  $4\times$  when full KV cache is offloaded to CPU. Since both the average summary lengths and consensus count  $c$  are fixed, the savings increase significantly with longer sequence length and larger batch sizes. As a tradeoff, due to the CPU-GPU data transfer latency, ZoomR

incurs a reduction in throughput (see Figure 3b), which is limited by the PCIe bandwidth. Therefore, ZoomR provides a major memory efficiency boost at the cost of reduced throughput, while maintaining strong performance compared to other KV cache compression methods.

### Consensus Dynamics in Thought Compression.

Our analysis reveals that consensus patterns between attention heads serve as a useful diagnostic signal for reasoning quality in mathematical tasks. We define an agreeability metric  $AG$ , as a measure of consensus among all attention heads regarding the importance of summary segments. Specifically, it quantifies the fraction of total “votes” that are assigned to the most frequently chosen summaries, i.e. the consensus set,  $AG = \sum_{i \in \mathcal{I}_c} v_i / \sum_{j \in \mathcal{I}_{\text{all}}} v_j$ . We observe a striking early advantage for correct answers, with 5.7% higher agreeability in the first 38 recompute steps compared to incorrect answers (see Figure 3d). This early consensus advantage suggests that correct reasoning converges quickly to a coherent solution path, while incorrect reasoning exhibits more exploratory behavior. The temporal dynamics show a clear crossover at step 38, after which incorrect answers demonstrate higher agreeability than correct ones. This pattern indicates that incorrect answers may require more extensive

exploration or get stuck on reasoning paths, unable to generate the correct answer.

Critically, incorrect answers show significantly higher variability in *AG* throughout the reasoning process, with a standard deviation of 14.82% compared to 7.87% for correct answers (see Figure 3e). This increased variability reflects the inherent instability of incorrect reasoning paths. This difference persists across all time windows, suggesting that consensus stability itself may be a fundamental property distinguishing correct from incorrect mathematical reasoning.

These findings support our hypothesis that ZoomR leverages consensus dynamics to guide reasoning quality through selective KV caching, offering new insights into how models can “self-assess” reasoning quality through internal agreement patterns during selective attention mechanisms.

**Ablation Studies.** We conduct ablation studies to assess the impact of the hyperparameters consensus count  $c$  and top- $k$ , on performance. Specifically, we run the experiments on MATH500 using the Llama model. With  $c = 1$ , only one summary is expanded to fine-details, and the performance drops by 3% compared to  $c = 2$ . However, for  $c \geq 2$ , there is only a minor improvement of 1%, while the number of KVs being selected increases by over 40%, as more summaries are being expanded (see Figure 3c). For  $k = 1$ , the performance is also noticeably worse, seeing close to a 3% drop compared to  $k = 2$ . This suggests that selecting only one top summary per head discards important context, reducing aggregation quality. However, when  $k \geq 2$ , the accuracy plateaus (see Figure 3f). The increase in number of tokens selected is also more gradual since increasing  $k$  only increases the number of summaries to select, which contribute much less to the KV cache.

## 5 Related Work

A central challenge in efficient reasoning is reducing both memory and compute overhead during inference. One line of work addresses this by reducing the number of tokens generated during inference (Sui et al., 2025). Latent reasoning and summarization based approaches enable efficient reasoning by summarizing the intermediate thinking steps. For instance, CODI and Coconut (Shen et al., 2025b; Hao et al., 2024b) train an LLM to compress a natural-language chain-of-thought into a continuous latent trajectory. LightThinker (Zhang

et al., 2025) trains an LLM to replace verbose intermediate reasoning with concise “gist” tokens and discards the full intermediate thoughts during decoding. While these methods offer inference-time efficiency, they rely on latent or opaque representations that are difficult to interpret or verify. Moreover, memory savings are typically a secondary concern. In contrast, our focus is on memory-centric techniques, specifically those involving KV cache compression.

**KV Cache Compression.** KV cache compression reduces both storage and the computational cost of LLM inference. KV cache quantization reduces memory by storing the key and value tensors with lower numerical precision (Zirui Liu et al., 2023; Hooper et al., 2025). In addition, KVs can also be compressed with low-rank projections along the model dimension (Saxena et al., 2024; Zhu et al., 2025b). These techniques are complementary to our method, as they operate on the representation level (e.g., precision or dimensionality).

Token selection methods retain or evict tokens based on learned or heuristic policies. StreamingLLM (Xiao et al., 2024) exploits the attention-sink effect to retain the initial tokens and the most recent context. SnapKV and H2O (Li et al., 2024; Zhang et al., 2023) rank and retain important tokens using scoring mechanisms. Quest, ShadowKV, and SentenceKV (Tang et al., 2024; Sun et al., 2025; Zhu et al., 2025a) move the KV cache generated during the prefill stage from GPU to CPU memory and retrieve a subset of KVs to attend to during decoding. While there is a growing literature of KV cache selection methods that enable memory efficiency for long context input, most of these approaches assume that the full KV cache is on GPU memory during the decoding stage, and are not designed for long-output reasoning tasks. Our work addresses this overlook scenario.

## 6 Conclusion

In this paper, we present ZoomR, a dynamic KV cache selection policy that enables memory efficient reasoning. By fine-tuning LLMs to generate summaries, and dynamically selecting summaries to “zoom” into based on a global consensus between attention heads, ZoomR enables efficient reasoning with higher fidelity. Extensive experiments show that ZoomR consistently outperforms other baseline approaches, maintaining competitive performance compared to full KV cache.

## Limitations

One key factor that can influence performance is quality of summaries used during data augmentation. In this work, we use summaries generated by the Llama-70B model, and do not evaluate the effect of using summaries generated by larger, more powerful models. The granularity of segment selection for summarization is fixed at the paragraph level. It remains unclear whether summarizing at finer or coarser granularities would improve or degrade ZoomR’s performance. Finally, ZoomR is primarily evaluated on mathematical reasoning tasks. Future work should explore its applicability to other domains such as code generation and creative writing, where reasoning structures may differ significantly.

## References

- Simon A. Aytes, Jinheon Baek, and Sung Ju Hwang. 2025. [Sketch-of-thought: Efficient llm reasoning with adaptive cognitive-inspired sketching](#). *Preprint*, arXiv:2503.05179.
- Frederic Charles Bartlett. 1932. *Remembering: A Study in Experimental and Social Psychology*. Cambridge University Press, Cambridge.
- Guoxuan Chen, Han Shi, Jiawei Li, Yihang Gao, Xiaozhe Ren, Yimeng Chen, Xin Jiang, Zhenguo Li, Weiyang Liu, and Chao Huang. 2025. [SepLLM: Accelerate large language models by compressing one segment into one separator](#). *Preprint*, arXiv:2412.12094.
- DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. [Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning](#). *Preprint*, arXiv:2501.12948.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. [The llama 3 herd of models](#). *Preprint*, arXiv:2407.21783.
- Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2025. [Token-budget-aware llm reasoning](#). *Preprint*, arXiv:2412.18547.
- Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2024a. [Training large language models to reason in a continuous latent space](#). *Preprint*, arXiv:2412.06769.
- Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2024b. [Training large language models to reason in a continuous latent space](#). *arXiv preprint arXiv:2412.06769*.
- Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2025. [Kvquant: Towards 10 million context length llm inference with kv cache quantization](#). *Preprint*, arXiv:2401.18079.
- Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. [Lora: Low-rank adaptation of large language models](#). *Preprint*, arXiv:2106.09685.
- Bespoke Labs. 2025. [Bespoke-stratos: The unreasonable effectiveness of reasoning distillation](#). <https://www.bespokelabs.ai/blog/bespoke-stratos-the-unreasonable-effectiveness-of-reasoning-distillation>. Accessed: 2025-01-22.
- Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. [Snapkv: Llm knows what you are looking for before generation](#). *Advances in Neural Information Processing Systems*, 37:22947–22970.
- OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, and 244 others. 2024a. [Openai o1 system card](#). *Preprint*, arXiv:2412.16720.
- OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024b. [Gpt-4 technical report](#). *Preprint*, arXiv:2303.08774.
- Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. [Qwen2.5 technical report](#). *Preprint*, arXiv:2412.15115.
- Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. 2024. [Eigen attention: Attention in low-rank space for kv cache compression](#). *arXiv preprint arXiv:2408.05646*.
- Xuan Shen, Yizhou Wang, Xiangxi Shi, Yanzhi Wang, Pu Zhao, and Jiuxiang Gu. 2025a. [Efficient reasoning with hidden thinking](#). *Preprint*, arXiv:2501.19201.

- Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. 2025b. [Codi: Compressing chain-of-thought into continuous space via self-distillation](#). *arXiv preprint arXiv:2502.21074*.
- Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, Hanjie Chen, and Xia Hu. 2025. [Stop overthinking: A survey on efficient reasoning for large language models](#). *Preprint*, arXiv:2503.16419.
- Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. 2025. [Shadowkv: Kv cache in shadows for high-throughput long-context llm inference](#). *Preprint*, arXiv:2410.21465.
- Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. [Quest: Query-aware sparsity for efficient long-context llm inference](#). *Preprint*, arXiv:2406.10774.
- Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, Chuning Tang, Congcong Wang, Dehao Zhang, Enming Yuan, Enzhe Lu, Fengxiang Tang, Flood Sung, Guangda Wei, Guokun Lai, and 77 others. 2025. [Kimi k1.5: Scaling reinforcement learning with llms](#). *Preprint*, arXiv:2501.12599.
- Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. 2025. [Tokenskip: Controllable chain-of-thought compression in llms](#). *Preprint*, arXiv:2502.12067.
- Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. [Efficient streaming language models with attention sinks](#). *Preprint*, arXiv:2309.17453.
- Yuchen Yan, Yongliang Shen, Yang Liu, Jin Jiang, Mengdi Zhang, Jian Shao, and Yueting Zhuang. 2025. [Infythink: Breaking the length limits of long-context reasoning in large language models](#). *Preprint*, arXiv:2503.06692.
- Jintian Zhang, Yuqi Zhu, Mengshu Sun, Yujie Luo, Shuofei Qiao, Lun Du, Da Zheng, Huajun Chen, and Ningyu Zhang. 2025. [Lightthinker: Thinking step-by-step compression](#). *Preprint*, arXiv:2502.15589.
- Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. [H<sub>2</sub>o: Heavy-hitter oracle for efficient generative inference of large language models](#). *Preprint*, arXiv:2306.14048.
- Yuxuan Zhu, Ali Falahati, David H. Yang, and Mohammad Mohammadi Amiri. 2025a. [Sentencekv: Efficient llm inference via sentence-level semantic kv caching](#). *Preprint*, arXiv:2504.00970.
- Yuxuan Zhu, David H. Yang, Mohammad Mohammadi Amiri, Keerthiram Murugesan, Tejaswini Pedapati, and Pin-Yu Chen. 2025b. [Ojakv: Context-aware online low-rank kv cache compression with oja's rule](#). *Preprint*, arXiv:2509.21623.
- 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 Appendix

### A.1 Summarization

ZoomR uses a Llama3-70B model to generate summaries and augment the bespoke-17K dataset. Specifically, we use the summarization prompt shown in Figure 4. Since summaries can be generated independently, this process can be greatly accelerated with the use of larger batch sizes and API calls. Once we collect all the summaries, these are then inserted back into the original dataset between special tokens `<|begin_of_summary|>` and `<|end_of_summary|>`, as illustrated in Figure 5. Fine-tuning hyperparameters are shown in Table 2.

Table 2: LoRA Fine-tuning Hyperparameters

Parameter	Value
LoRA Rank ( $r$ )	16
LoRA Alpha ( $\alpha$ )	32
Number of Epochs	3
Learning Rate	$1 \times 10^{-4}$
Batch Size	1
Gradient Accumulation Steps	4
Maximum Sequence Length	16,384

### A.2 Algorithm Details

Algorithm 1 shows the full algorithm for ZoomR.

### A.3 Semantic Boundaries

In our methodology, the semantic boundaries are determined by a preset list of punctuation tokens. The specific tokens used are `.`, `!`, `?`, `...`, `:`, `\n\n`, and `\n\n\n`. The average recompute interval occurs after approximately 35 tokens, which matches the expected sentence length.

### A.4 ZoomR Generated Example

ZoomR generates summaries dynamically during reasoning. Figure 6 shows part of the reasoning thoughts from a challenging AIME2025 problem.

### Summarization Prompt

System Prompt: You are a reasoning trace compressor. Your goal is to compress intermediate reasoning steps into minimal but faithful summaries.

Instruction:

"You are compressing a single paragraph from a chain-of-thought reasoning"  
"trace into a short summary token sequence.\ Use the prior context to"  
"understand what the current step is doing, but summarize only the"  
"current paragraph.\ Context: " + {context} + "\n"  
"Compress the following paragraph into a single short phrase (10-12 tokens)"  
"that captures the key logical conclusion or operation performed."  
"Prioritize preserving: mathematical results, logical deductions, and"  
"intermediate conclusions.\ You may omit redundant reasoning steps."  
"Respond in JSON with the compressed summary in 'results'.\n"  
"Paragraph: \n" + {paragraph}

Figure 4: Prompt used to generate summaries

---

### Algorithm 1 ZoomR

---

1: **Require:** Prompt indices  $\mathcal{I}_p$ , per-head top- $k$ , consensus count  $c$ , recent window size  $w$ .

2: **Define:** Regular segments  $\{\mathcal{R}_i\}$  and summary segments  $\{\mathcal{S}_i\}$ .

**Prefill:** Initialize KV cache using prompt  $P$ .

**Decoding:**

3: **for** each step  $t = 1, \dots, N_g$  **do**

4:   **for** each layer  $l = 1, \dots, N_L$  **do**

▷ Per-head importance scoring and voting

5:     **for** each head  $h = 1, \dots, H$  **do**

6:       Append  $\mathbf{k}_t^{(l,h)}$  and  $\mathbf{v}_t^{(l,h)}$  to KV cache

7:       **if** a new summary  $\mathcal{S}_{N_t}$  is complete **then**

8:           $\bar{\mathbf{k}}_{N_t}^{(l,h)} \leftarrow \frac{1}{|\mathcal{S}_{N_t}|} \sum_{j \in \mathcal{S}_{N_t}} \mathbf{k}_j^{(l,h)}$ .

▷ Compute mean summary key

9:       **end if**

10:        $\alpha_i^{(l,h)} \leftarrow (\mathbf{q}_t^{(l,h)})^\top \bar{\mathbf{k}}_i^{(l,h)}$  for all  $i$ .

11:        $\mathcal{I}_{\text{top-k}}^{(l,h)} \leftarrow \arg \text{top-k}(\alpha_i^{(l,h)})$ .

12:     **end for**

13:   **end for**

14:   **if** at a semantic boundary **then**

15:      $\mathcal{I}_{\text{all}} \leftarrow \bigcup_{l,h} \mathcal{I}_{\text{top-k}}^{(l,h)}$ .

▷ Global aggregation and consensus

16:     For each  $i \in \mathcal{I}_{\text{all}}$ , compute  $v_i = \sum_{l,h} \mathbb{I}(i \in \mathcal{I}_{\text{top-k}}^{(l,h)})$ .

17:      $\mathcal{I}_c \leftarrow \arg \text{top-c}(v_i)$ .

18:      $\mathcal{I}_s \leftarrow \mathcal{I}_{\text{all}} \setminus \mathcal{I}_c$ .

19:      $\mathcal{I}_w \leftarrow \{t - w + 1, \dots, t - 1\}$ .

20:      $\mathcal{I}_f \leftarrow \mathcal{I}_p \cup \mathcal{I}_w \cup \left( \bigcup_{i \in \mathcal{I}_c} \mathcal{R}_i \right) \cup \left( \bigcup_{i \in \mathcal{I}_s} \mathcal{S}_i \right)$ .

▷ Construct final multi-granularity context

21:   **end if**

22: **end for**

---

### Augmented Dataset Example

**Question:** Return your final response within  $\boxed{\quad}$ . A quadrilateral has vertices at (0,1), (3,4), (4,3) and (3,0). Its perimeter can be expressed in the form  $a\sqrt{2} + b\sqrt{10}$  with  $a$  and  $b$  integers. What is the sum of  $a$  and  $b$ ?

**Thoughts:** Okay, so I have this problem where I need to find the perimeter of a quadrilateral with vertices at (0,1), (3,4), (4,3), and (3,0). The perimeter is supposed to be expressed in the form  $a\sqrt{2} + b\sqrt{10}$ , where  $a$  and  $b$  are integers, and then I have to find the sum of  $a$  and  $b$ . Alright, let me start by understanding how to approach this.

First, the perimeter is the sum of the lengths of all four sides. I need to find the distance between each pair of consecutive vertices. The distance formula between two points  $(x_1, y_1)$  and  $(x_2, y_2)$  is  $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ .

Let the vertices be A(0,1), B(3,4), C(4,3), D(3,0). The sides are AB, BC, CD, and DA.

Perimeter =  $AB + BC + CD + DA$

I'll calculate each side one by one.

Starting with AB: from (0,1) to (3,4). The length AB is  $\sqrt{(3 - 0)^2 + (4 - 1)^2} = \sqrt{3^2 + 3^2} = \sqrt{9 + 9} = \sqrt{18}$ .

This can be simplified:  $\sqrt{18} = \sqrt{9 \times 2} = 3\sqrt{2}$ . So AB is  $3\sqrt{2}$ .

Next, BC: from (3,4) to (4,3). The length BC is  $\sqrt{(4 - 3)^2 + (3 - 4)^2} = \sqrt{1^2 + (-1)^2} = \sqrt{1 + 1} = \sqrt{2}$ .

$AB = 3\sqrt{2}, BC = \sqrt{2}$

Third side is CD: from (4,3) to (3,0). The length CD is  $\sqrt{(3 - 4)^2 + (0 - 3)^2} = \sqrt{(-1)^2 + (-3)^2} = \sqrt{1 + 9} = \sqrt{10}$ .

Last side is DA: from (3,0) back to (0,1). The length DA is  $\sqrt{(0 - 3)^2 + (1 - 0)^2} = \sqrt{(-3)^2 + 1^2} = \sqrt{9 + 1} = \sqrt{10}$ .

Now, let's add up all these lengths:  $AB + BC + CD + DA = 3\sqrt{2} + \sqrt{2} + \sqrt{10} + \sqrt{10}$ .

$AB = 3\sqrt{2}, BC = \sqrt{2}, CD = \sqrt{10}, DA = \sqrt{10}$

Combining like terms:  $(3\sqrt{2} + \sqrt{2}) + (\sqrt{10} + \sqrt{10}) = 4\sqrt{2} + 2\sqrt{10}$ .

So the total perimeter is  $4\sqrt{2} + 2\sqrt{10}$ .

...

Figure 5: An example of a segment of reasoning process with intermediate summaries.

### ZoomR Example

**Question:** The 9 members of a baseball team went to an ice cream parlor after their game. Each player had a single-scoop cone of chocolate, vanilla, or strawberry ice cream. At least one player chose each flavor, and the number of players who chose chocolate was greater than the number of players who chose vanilla, which was greater than the number of players who chose strawberry. Let  $N$  be the number of different assignments of flavors to players that meet these conditions. Find the remainder when  $N$  is divided by 1000.

**Thoughts:** Okay, so I need to solve this problem about assigning ice cream flavors to a baseball team. Let me read the problem again carefully to make sure I understand all the conditions.

There are 9 team members, each getting a single scoop of chocolate, vanilla, or strawberry ice cream. The conditions are:

1. Each flavor is chosen by at least one player.
2. The number of players who chose chocolate is greater than those who chose vanilla, which in turn is greater than those who chose strawberry. So,  $Ch > Van > Str$ .

We need to find the number of different assignments (flavorings) that meet these conditions, then find  $N \pmod{1000}$ .

Hmm. Let me break this down. So, each player has 3 choices:  $C$ ,  $V$ , or  $S$ . But with constraints on the counts. The key constraints are:

- All three flavors are used at least once.
- The counts satisfy  $Ch > Van > Str$ .

So, variables:

Let  $C$  = number of chocolate,  $V$  = vanilla,  $S$  = strawberry.

Given that  $C > V > S$ , and  $C + V + S = 9$ . Also, each of  $C, V, S \geq 1$ . We need to find the number of ordered triples  $(C, V, S)$  that satisfy these conditions, and then multiply by the number of ways to assign these counts to the 9 players, considering the different permutations.

Wait, no. Actually, the assignments are different distributions of the flavors to the players. So, for each valid combination  $(C, V, S)$ , the number of ways is the multinomial coefficient:  $\frac{9!}{C!V!S!}$ . Then, sum this over all valid triples  $(C, V, S)$  where  $C > V > S \geq 1$  and  $C + V + S = 9$ .

But first, we need to find all possible triples  $(C, V, S)$  that satisfy the inequalities and sum to 9. Let me first figure out all possible triples  $(C, V, S)$  with  $C > V > S \geq 1$  and  $C + V + S = 9$ . Since  $S \geq 1$ ,  $V \geq 1$ ,  $C \geq 1$ , and  $C > V > S$ .

Let me start by considering possible values for  $S$ . Since  $S$  must be at least 1, and  $V$  must be at least  $S + 1$ , and  $C$  must be at least  $V + 1$ . So,  $S$  can be from 1 up to some maximum value where these inequalities can still hold.

Let me try to find the possible values of  $S$ .  $S \geq 1$ ,  $V \geq S + 1$ ,  $C \geq V + 1$ . Let's denote  $S = k$ . Then,  $V$  must be at least  $k + 1$ , and  $C$  must be at least  $V + 1 = k + 2$ . So,  $C \geq k + 2$ .

...

Thus, the remainder when  $N$  is divided by 1000 is 16.

Figure 6: An example from AIME2025. ZoomR generated thoughts with summarization.