

Sparse-RL: Breaking the Memory Wall in LLM Reinforcement Learning via Stable Sparse Rollouts

Sijia Luo^{1,2*}, Xiaokang Zhang^{1,2*}, Yuxuan Hu^{1,2}, Bohan Zhang^{1,2},
Ke Wang^{1,2}, Jinbo Su^{1,2}, Mengshu Sun⁴, Lei Liang⁴, Jing Zhang^{1,3†}

¹School of Information, Renmin University of China, Beijing, China

²Key Laboratory of Data Engineering and Knowledge Engineering, Beijing, China

³Engineering Research Center of Database and Business Intelligence, Beijing, China

⁴Ant Group, Hangzhou, China

sijialuo23@gmail.com, {zhang2718, zhang-jing}@ruc.edu.cn

Abstract

Reinforcement Learning (RL) has become essential for eliciting complex reasoning capabilities in Large Language Models (LLMs). However, the substantial memory overhead of storing Key-Value (KV) caches during long-horizon rollouts acts as a critical bottleneck, often prohibiting efficient training on limited hardware. While existing KV compression techniques offer a remedy for inference, directly applying them to RL training induces a severe policy mismatch, leading to catastrophic performance collapse. To address this, we introduce Sparse-RL, which empowers stable RL training under sparse rollouts. We show that instability arises from a fundamental policy mismatch among the dense old policy, the sparse sampler policy, and the learner policy. To mitigate this issue, Sparse-RL incorporates Sparsity-Aware Rejection Sampling and Importance-based Reweighting to correct the off-policy bias introduced by compression-induced information loss. Experimental results show that Sparse-RL reduces rollout overhead compared to dense baselines while preserving the performance. Furthermore, Sparse-RL inherently implements sparsity-aware training, significantly enhancing model robustness during sparse inference deployment. The corresponding training data and code are publicly available on the repository¹.

1 Introduction

Reasoning large models, such as OpenAI-o1 (Jaech et al., 2024) and DeepSeek-R1 (Guo et al., 2025), have demonstrated exceptional capabilities in tackling complex reasoning tasks such as mathematical reasoning and code generation (Chen et al., 2025; Liu and Zhang, 2025). Reinforcement learning (RL) has emerged as an effective post-training

paradigm for enhancing LLMs’ strong reasoning capabilities (Ouyang et al., 2022; Shao et al., 2024). By incentivizing models to explore extended chains of reasoning and optimizing them toward explicit behavioral goals, RL enables LLMs to better align with human preferences and execute tasks requiring structured reasoning and verifiable decision-making (Guo et al., 2025; Team, 2025).

The remarkable effectiveness of RL training comes at an extremely high computational cost. Modern RL training primarily consists of two stages: rollout and training, with the rollout stage dominating the overall time in RL execution. Recent studies (Shao et al., 2025; Wang et al., 2025) show that the rollout stage accounts for approximately 70% of the total time. Simultaneously, this large-scale generation faces a severe “memory wall”: as the sequence lengths increase, the continuously expanding KV cache consumes substantial GPU memory (Zhao et al., 2025). To prevent out-of-memory (OOM) errors during long-tail sample generation, rollout batch sizes must be constrained. This limitation reduces GPU utilization and constrains the throughput of scalable RL training.

To alleviate computational and memory bottlenecks in RL training, KV cache compression techniques offer a viable solution—originally designed to accelerate long-context inference (Zhang et al., 2023b). By selectively retaining key-value pairs (typically identified through cumulative attention scores), representative compression algorithms (e.g., H2O (Zhang et al., 2023b), StreamingLLM (Xiao et al., 2023), SnapKV (Li et al., 2024)) can effectively reduce memory complexity from linear growth to a fixed token budget, decoupling memory consumption from sequence length. Given that existing training-free compression methods demonstrate strong performance in inference scenarios, a natural question emerges: *Can these training-free compression methods be directly applied to the rollout stage of RL train-*

*Equal Contributions.

†Corresponding Author.

¹<https://github.com/RUCKBReasoning/Sparse-RL>

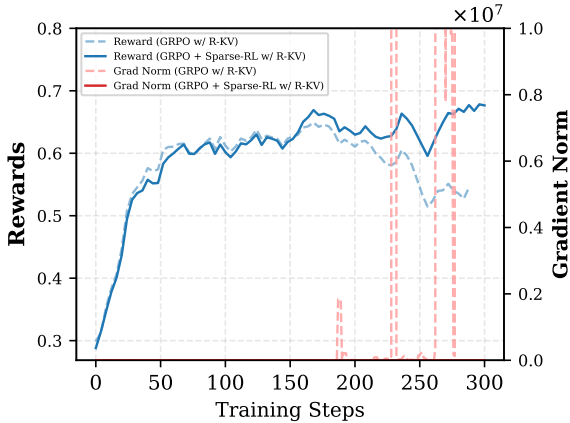


Figure 1: Impact of Sparse-RL on training stability. We compare training dynamics between naive GRPO and Sparse-RL during Qwen2.5-7B training with R-KV compression. While naive GRPO suffers from a reward collapse and gradient spikes, Sparse-RL maintains stable reward growth with gradient norms remaining consistently low (appearing near the x-axis relative to the massive spikes of the naive baseline).

ing? Theoretically, this could eliminate the aforementioned long-tail memory shortage risk, thereby supporting larger batch sizes to accelerate rollout.

However, directly applying compression methods designed for static inference to dynamic RL training is non-trivial and faces the following challenges: (1) *Policy Mismatch*: In standard PPO (Schulman et al., 2017) or GRPO (Shao et al., 2024), sampling and training policies typically originate from identical distributions. In this setup, responses generated during sampling rely on a sparse policy based on compressed key-value pairs, while gradient updates utilize a dense policy. This mismatch introduces a fundamental off-policy issue (Yao et al., 2025; Liu et al., 2025). (2) *Anomalous Sample*: Existing KV compression algorithms are primarily designed for long-context inference and do not account for the randomness inherent in the RL exploration stage. During sampling, the information loss induced by compression may generate anomalous samples such as infinite repetition (example in Appendix G), which can produce massive negative gradients or noise, undermining the stability of RL training. Crucially, these two effects jointly result in catastrophic training collapse. The policy mismatch amplifies the influence of corrupted trajectories, while anomalous samples introduce extreme gradients and high-variance updates. As observed in Figure 1, after training for about 200 steps, the rewards of the naive GRPO

with R-KV (Cai et al., 2025) compression begin to decrease and the gradient norm spikes at the same time.

To address these challenges, we propose Sparse-RL, a novel framework that enables stable RL training with sparse rollouts through KV cache compression. For anomalous samples generated due to KV compression, we design Sparsity-Aware Rejection Sampling, which utilizes the token probability ratio between sparse and dense policies as a detector to eliminate anomalous sequence samples. To further correct the policy mismatch within valid samples, we use Importance-based Reweighting which leverage the token probability ratio between sparse and dense policies to reweight the policy gradient, recovering unbiased estimates for the dense learner. This design enables Sparse-RL to theoretically achieve stable, efficient RL training under relatively low memory budgets. Since its correction mechanism relies solely on probability distribution rather than specific compression operators, Sparse-RL adapts to various mainstream training-free compression algorithms, providing the community with a universal, memory-efficient RL paradigm.

Our contributions can be summarized as follows: 1) We pioneer the exploration of integrating training-free KV Cache compression technology into the rollout phase of RL training. 2) We identified and resolved training collapse caused by policy mismatch resulting from sparse rollouts. We propose Sparsity-Aware Rejection Sampling to filter anomalous trajectories introduced by KV compression, and Importance-based Reweighting to ensure stable policy updates. 3) We evaluate Sparse-RL on 7 benchmarks with 4 models and 2 KV compression methods. Under a fixed token budget, Sparse-RL matches dense rollouts and even outperforms them on Qwen2.5-1.5B. Up to 7B scale, it retains over 96% of dense performance. 4) We further observe that Sparse-RL inherently implements *Sparsity-Aware Training*, significantly enhancing model robustness during sparse inference deployment in downstream tasks. This successfully bridges the gap between efficient training and efficient deployment.

2 Related Work

Efficient Reinforcement Learning for LLM. Reinforcement Learning (RL) has emerged as the cornerstone for eliciting complex reasoning capabilities in LLMs, as demonstrated by OpenAI-

o1 (Jaech et al., 2024) and DeepSeek-R1 (Guo et al., 2025). Unlike offline alignment methods such as DPO (Rafailov et al., 2023), which are data-efficient but lack exploratory capacity, online algorithms like PPO (Schulman et al., 2017) and GRPO (Shao et al., 2024) are essential for discovering novel reasoning paths. Compared to PPO, GRPO eliminates the critic by estimating advantages through group-wise ranking, significantly reducing memory consumption while maintaining strong performance on verifiable reasoning tasks. GSPO (Zheng et al., 2025) elevates the optimization granularity from token-level to sequence-level, rectifying the credit assignment ambiguity inherent in dense rewards. From a system perspective, frameworks like Seer (Qin et al., 2025), AgentRL (Zhang et al., 2025a) and AReal (Fu et al., 2025) propose asynchronous architectures that decouple trajectory generation from policy learning, significantly improve training throughput (Zhang et al., 2025c,b).

KV Cache Compression. Existing methods primarily include token eviction (Ge et al., 2023; Liu et al., 2023; Li et al., 2024), merging (Zhang et al., 2024), quantization (Hooper et al., 2024; Liu et al., 2024), and low-rank decomposition (Sun et al., 2024). For reasoning models, we consider dynamic eviction strategies that continuously evicts key-value caches during decoding. Early methods like StreamingLLM (Xiao et al., 2023) use sliding windows combined with attention sinks. SnapKV (Li et al., 2024), PyramidKV (Cai et al., 2024) and Ada-KV (Feng et al., 2024) prune tokens based on attention scores. More recently, R-KV (Cai et al., 2025) targets redundant tokens common in reasoning chains. By eliminating redundancy via similarity clustering during both pre-filling and decoding, R-KV achieves state-of-the-art results on MATH500 (Hendrycks et al., 2021) and AIME24. However, all aforementioned methods are designed for inference, and their application to RL training dynamics has not yet been investigated.

3 Problem Definition

In the context of RL training with KV cache compression, we aim to optimize a policy π_θ to maximize the expected objective. Due to sparse rollout generation and policy staleness, three distinct policy distributions coexist during training, giving rise to an intrinsic policy mismatch.

Notation We define an autoregressive LLM param-

eterized by θ as a policy π_θ . We use x to denote an input prompt and D as the prompt set. Under the policy π_θ , the likelihood of a response o to a prompt x is denoted as $\pi_\theta(o|x) = \prod_{t=1}^{|o|} \pi_\theta(o_t|x, o_{<t})$ where $|o|$ is the number of tokens in o .

Dense Old Policy represents the ideal and uncompressed policy derived from the full context history $o_{<t}$:

$$\pi_{\theta_{\text{old}}}(o_t|x, o_{<t}) = P(o_t|x, o_{<t}; \theta_{\text{old}}) \quad (1)$$

Sparse Sampler Policy is the policy actually used for generating responses, emerging from the sparse version of the dense old policy. Due to the use of the KV compression operator $\mathcal{M}(\cdot)$, the conditional probability at time step t depends on compressed historical information:

$$\pi_{\theta_{\text{sparse}}}(o_t|x, o_{<t}) = P(o_t|x, \mathcal{M}(o_{<t}); \theta_{\text{old}}) \quad (2)$$

Note that while we denote it as $\pi_{\theta_{\text{sparse}}}$ for notational symmetry, it does not possess a distinct set of parameters; rather, it represents the sparse view of $\pi_{\theta_{\text{old}}}$ under compression.

Learner Policy is the policy currently being optimized, typically diverges from θ_{old} due to policy staleness:

$$\pi_\theta(o_t|x, o_{<t}) = P(o_t|x, o_{<t}; \theta) \quad (3)$$

4 Methodology

To address the policy mismatch introduced by KV compression, we first analyze the discrepancy from an Importance Sampling (IS) perspective. Guided by this derivation, we propose Sparse-RL, which utilizes Sparsity-Aware Rejection Sampling to filter anomalous trajectories and Importance-based Reweighting to recover unbiased gradient estimates.

4.1 The Decomposition of Policy Mismatch

Standard on-policy RL algorithms (e.g., PPO) assume the sampler policy matches the learner policy. In our setting, sampling from $\pi_{\theta_{\text{sparse}}}$ introduces a structural bias. To obtain an unbiased gradient estimate for π_θ , one would theoretically employ the total importance sampling (IS) weight $w_{\text{total}} = \frac{\pi_\theta}{\pi_{\theta_{\text{sparse}}}}$.

We observe that the discrepancy between the learner and the sparse sampler policy stems from two distinct sources: the difference between dense

and sparse policy distributions caused by information loss during compression, and the standard off-policy bias arising from optimizing new policy based on empirical data generated by old policy. Therefore, the importance weights can be decomposed as follows:

$$\frac{\pi_\theta(o_t|x, o_{<t})}{\pi_{\theta_{\text{sparse}}}(o_t|x, o_{<t})} = \underbrace{\frac{\pi_\theta(o_t|x, o_{<t})}{\pi_{\theta_{\text{old}}}(o_t|x, o_{<t})}}_{\text{policy staleness}} \times \underbrace{\frac{\pi_{\theta_{\text{old}}}(o_t|x, o_{<t})}{\pi_{\theta_{\text{sparse}}}(o_t|x, o_{<t})}}_{\text{sparsity-induced mismatch}} \quad (4)$$

4.2 Sparsity-Aware Rejection Sampling

Quantifying Distributional Discrepancy. KV compression can be viewed as a lossy approximation operator $\mathcal{M}(\cdot)$ on the historical state $o_{<t}$. While often accurate, this approximation causes $\pi_{\theta_{\text{sparse}}}$ to occasionally assign high probability to tokens that are statistically impossible under the full-context $\pi_{\theta_{\text{old}}}$, leading to abnormal sequences. To detect these trajectories, we define the sparsity consistency ratio ξ_t at each time step t :

$$\xi_t = \frac{\pi_{\theta_{\text{old}}}(o_t|x, o_{<t})}{\pi_{\theta_{\text{sparse}}}(o_t|x, o_{<t})} \quad (5)$$

A value of $\xi_t \approx 1$ implies local consistency between sparse policy and dense policy. Conversely, $\xi_t \rightarrow 0$ signals a support mismatch, where the sparse policy enters a region of the state space unsupported by the dense model.

Rejection Sampling. In reasoning tasks, a single hallucinated step can invalidate the entire chain of thought. Therefore, we use a strict sequence-level validity constraint: a trajectory is accepted if and only if every generated token remains within the support of the dense policy. We define the sequence-level rejection weight $M^{\text{RS}}(o)$ as:

$$M^{\text{RS}}(o) = \begin{cases} 0 & \text{if } \exists t \in [1, |o|] \text{ s.t. } \xi_t < \epsilon \\ 1 & \text{otherwise} \end{cases} \quad (6)$$

where ϵ is a threshold (e.g., $1e-4$). This implies that if the sparse policy generates a single anomalous token, the weight of the entire sequence is zeroed out. As a result, the optimizer computes gradients only from logically valid trajectories that remain consistent with the behavior of the dense policy.

4.3 Importance-based Reweighting

As Sparse-RL is built upon GRPO (preliminaries detailed in Appendix D), the policy model samples a group of G outputs $\{o_1, \dots, o_G\}$ from the sparse

policy $\pi_{\theta_{\text{sparse}}}$ for a given prompt x . After filtering out hallucinated trajectories through rejection sampling, a distributional bias remains: the valid samples are drawn from $\pi_{\theta_{\text{sparse}}}$, but we aim to estimate expectations under the dense policy $\pi_{\theta_{\text{old}}}$. To rigorously correct this bias, we incorporate the sparsity consistency ratio $\xi_{i,t} = \frac{\pi_{\theta_{\text{old}}}(o_t|x, o_{i,<t})}{\pi_{\theta_{\text{sparse}}}(o_t|x, o_{i,<t})}$ as a reweighting factor, and $w_{i,t}(\theta) = \frac{\pi_\theta(o_t|x, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_t|x, o_{i,<t})}$ denote the ratio of policy staleness.

The objective function of Sparse-RL is formulated as:

$$\mathcal{J}_{\text{Sparse-RL}}(\theta) = \mathbb{E}_{\substack{x \sim \mathcal{D} \\ o_i \sim \pi_{\theta_{\text{sparse}}}}} \left[\frac{1}{G} \sum_{i=1}^G M^{\text{RS}}(o_i) \cdot \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \xi_{i,t} \cdot \min \left(w_{i,t}(\theta) \hat{A}_i, \text{clip}(w_{i,t}(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_i \right) \right] \quad (7)$$

where $M^{\text{RS}}(o)$ zeros out entire anomalous trajectories. The sparsity consistency ratio $\xi_{i,t}$ is applied outside the clipping operator to unbiasedly correct the distribution shift via importance sampling, while the clipping is restricted to $w_{i,t}(\theta)$ to enforce the trust region constraint relative to the dense policy.

Gradient Analysis For the Sparse-RL objective (Eq. 7), the policy gradient $\nabla_\theta \mathcal{J}_{\text{Sparse-RL}}(\theta)$ can be derived as follows (clipping is omitted for brevity):

$$\nabla_\theta \mathcal{J}_{\text{Sparse-RL}} = \mathbb{E}_{\substack{x \sim \mathcal{D} \\ o_i \sim \pi_{\theta_{\text{sparse}}}}} \left[\frac{1}{G} \sum_{i=1}^G M^{\text{RS}}(o_i) \cdot \frac{\hat{A}_i}{|o_i|} \cdot \sum_{t=1}^{|o_i|} \left(\xi_{i,t} \cdot w_{i,t}(\theta) \nabla_\theta \log \pi_\theta(o_t|x, o_{i,<t}) \right) \right] \quad (8)$$

If we use $\mathcal{V} = \{i \mid M^{\text{RS}}(o_i) = 1\}$ to denote the set of indices for valid trajectories that passed the rejection sampling filter, the gradient becomes:

$$\nabla_\theta \mathcal{J}_{\text{Sparse-RL}} = \mathbb{E}_{\substack{x \sim \mathcal{D} \\ o_i \sim \pi_{\theta_{\text{sparse}}} \\ i \in \mathcal{V}}} \left[\frac{1}{G} \sum_{i \in \mathcal{V}} \frac{1}{|o_i|} \cdot \hat{A}_i \cdot \sum_{t=1}^{|o_i|} \left(\xi_{i,t} \cdot w_{i,t}(\theta) \nabla_\theta \log \pi_\theta(o_t|x, o_{i,<t}) \right) \right] \quad (9)$$

The detailed derivation is provided in Appendix E.

Mechanistic Interpretation. The resulting gradient reveals a dual-level correction mechanism:

- **Sequence-level Filtering (M^{RS}):** The summation $\sum_{i \in \mathcal{V}}$ effectively removes any output o_i that contains hallucinations or abnormal reasoning steps caused by information loss. This ensures the advantage estimates \hat{A}_i are not corrupted by out-of-distribution samples.

Model	Rollout Method	GSM8K	MATH500	Gaokao	Minerva Math	Olympiad	AIME24	AMC23	Avg.	Toks. saving
Llama-3.2-1B-Instruct										
Base	–	36.2	22.8	17.9	5.5	5.5	1.2	8.8	14.0	-
GRPO	Dense	51.2	<u>33.6</u>	28.8	8.8	8.7	<u>2.9</u>	12.8	<u>21.0</u>	-
GRPO		49.8	30.0	23.9	5.5	7.6	1.7	12.8	18.8	35.1%
↔ + Sparse-RL	w/ R-KV	49.1	30.2	26.5	5.5	8.3	2.1	15.2	19.6	
GRPO		<u>52.2</u>	<u>33.6</u>	26.5	5.5	9.0	2.9	<u>13.8</u>	20.5	35.1%
↔ + Sparse-RL	w/ SnapKV	54.6	34.4	<u>27.0</u>	<u>6.2</u>	9.3	3.0	13.6	21.2	
Qwen2.5-1.5B										
Base	–	43.5	21.0	18.4	4.0	4.0	0.3	7.5	14.1	-
GRPO	Dense	72.9	58.0	46.5	16.9	<u>21.6</u>	3.6	28.5	35.4	-
GRPO		48.8	34.0	32.5	7.0	11.6	1.0	21.2	22.3	53.3%
↔ + Sparse-RL	w/ R-KV	75.3	55.8	<u>48.1</u>	20.2	21.9	2.9	<u>29.5</u>	36.2	
GRPO		66.3	37.6	33.8	8.1	16.1	3.1	18.0	26.1	53.3%
↔ + Sparse-RL	w/ SnapKV	<u>73.7</u>	<u>57.6</u>	48.8	<u>18.0</u>	20.7	<u>3.4</u>	<u>29.5</u>	<u>36.0</u>	
Qwen2.5-3B										
Base	–	76.0	55.8	45.2	19.9	21.6	4.1	27.3	35.7	-
GRPO	Dense	<u>83.5</u>	<u>64.2</u>	56.9	25.0	29.8	6.0	<u>38.6</u>	43.4	-
GRPO		79.0	56.6	46.2	21.7	23.0	4.7	27.0	36.9	42.0%
↔ + Sparse-RL	w/ R-KV	84.6	65.6	53.5	26.5	<u>27.7</u>	4.6	37.4	<u>42.8</u>	
GRPO		79.0	54.2	48.3	19.5	23.6	5.2	30.5	37.2	42.0%
↔ + Sparse-RL	w/ SnapKV	83.4	64.0	<u>54.8</u>	<u>25.4</u>	25.9	<u>5.3</u>	39.1	42.6	
Qwen2.5-7B										
Base	–	81.6	57.4	52.2	22.1	26.4	7.3	35.5	40.4	-
GRPO	Dense	<u>91.0</u>	73.8	63.9	37.1	39.1	14.5	<u>52.5</u>	53.1	-
GRPO		88.0	64.2	57.1	23.2	30.7	8.8	43.8	45.1	39.4%
↔ + Sparse-RL	w/ R-KV	91.5	<u>72.0</u>	<u>61.8</u>	<u>32.7</u>	<u>36.6</u>	<u>12.5</u>	52.7	<u>51.4</u>	
GRPO		73.4	54.6	46.0	16.9	19.0	2.6	25.2	34.0	39.4%
↔ + Sparse-RL	w/ SnapKV	90.1	71.4	60.8	26.8	32.9	10.2	47.6	48.5	

Table 1: Main results on 7 mathematical reasoning benchmarks. We compare our Sparse-RL framework (instantiated with R-KV and SnapKV) against the uncompressed Dense-GRPO baseline and the Naive sparse rollout baseline. The “Toks. saving” column indicates the reduction in KV cache storage compared to the dense rollout baseline during training. The top two performances are highlighted in **bold** and underlined.

- **Token-level Reweighting ($\xi_{i,t}$):** For the accepted trajectories, the token-level gradient is reweighted by $\xi_{i,t}$ to eliminate the distribution gap between sparse and dense policy.

5 Experiments

5.1 Experimental Setup

Backbone Models We conduct zero RL training experiments on Qwen2.5 (1.5B/3B/7B) (Qwen et al., 2025) and Llama-3.2-1B-Instruct (Dubey et al., 2024), to cover various model sizes and verify cross-architecture generalization.

Training Datasets Our experiments use the SimpleRL-Zoo (Zeng et al., 2025) dataset, which integrates the training sets of GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021). This

dataset is chosen based on the observation that, when initiating ZeroRL directly from a base model, successful training critically depends on using data that matches the model’s capability. The dataset is divided into three difficulty levels: Easy (GSM8K and MATH level 1), Medium (MATH levels 1–4), and Hard (MATH levels 3–5), with each split containing approximately 8,000 problems. Given the strong mathematical capabilities of the Qwen2.5 series and Llama3.2 instruct models, we use the hard level split as our training dataset.

Evaluation We evaluate the performance of our method on 7 standard mathematical reasoning benchmarks: GSM8K (Cobbe et al., 2021), MATH500 (Hendrycks et al., 2021), Gaokao (Zhang et al., 2023a), Min-

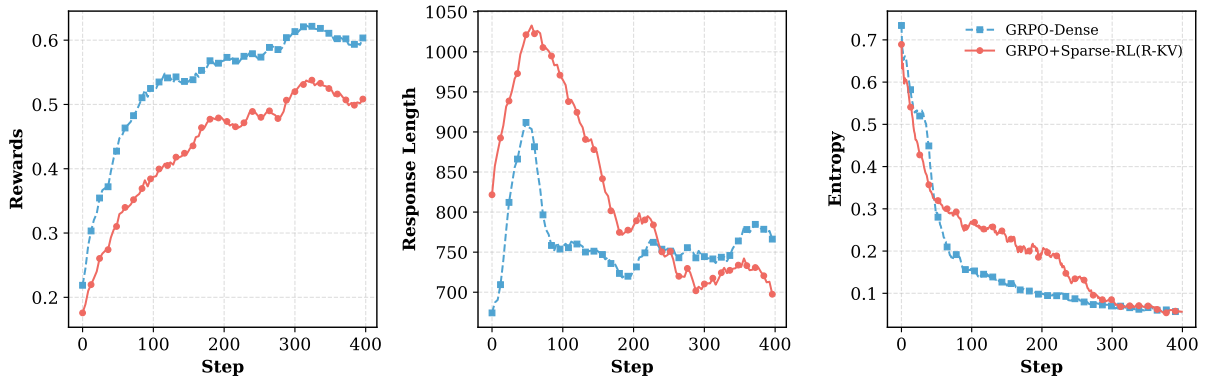


Figure 2: Comparison of training curves between GRPO-Dense and GRPO + Sparse-RL (with R-KV) on Qwen2.5-3B over 400 training steps. **Left:** Average rewards per step. **Middle:** Average response length (tokens). **Right:** Policy entropy.

erva Math (Lewkowycz et al., 2022), Olympiad-Bench (He et al., 2024), AIME24² and AMC23³. For AIME24 and AMC23, we report Avg@32: for each problem, we sample 32 responses and calculate the mean accuracy among these 32 responses; the final score is the average of the mean accuracy for each item over the dataset. For the other six benchmarks, we report Pass@1: generating one response per problem.

Baselines To assess the efficacy of our proposed framework, we design a comparative study involving three primary configurations:

- **Base:** The base model without RL training.
- **GRPO (Dense Rollout):** The standard GRPO algorithm using full KV cache rollouts without any compression. This serves as the dense baseline to measure the performance gap caused by sparse approximation.
- **GRPO (Naive Sparse Rollout):** We directly apply the KV cache compression methods (specifically R-KV and SnapKV) during the rollout generation phase of GRPO, devoid of off-policy correction mechanisms. This baseline highlights the catastrophic performance collapse arising from the policy mismatch.
- **GRPO + Sparse-RL (Ours):** We instantiate our Sparse-RL framework with two distinct compression algorithms: **Sparse-RL + SnapKV** (representing attention-based compression) and **Sparse-RL + R-KV** (representing redundancy-aware compression).

Implementation Details Our training is based on the slime framework (Zhu et al., 2025). We adopt a strict binary reward scheme, assigning a reward of 1 to correct responses and 0 otherwise. For each prompt, we sample $G = 8$ rollouts with a temperature of 1.0, top-p of 1.0, and a maximum response length of 4096 tokens. The global rollout batch size is set to 1024, with an update batch size of 256. The learning rate is set to 1e-6 with a KL loss coefficient of 1e-4. The ratio of rejection sampling is set to 1e-4. As for compression parameters, we set the KV budget to 512, which means during the rollout process, only 512 tokens are retained when generating each response. Experiments fine-tune backbone models for 400 training steps (≈ 1.5 epochs) on $4 \times$ NVIDIA H20-HGX-141GB GPUs.

5.2 Main Results

Table 1 presents the experimental results across 7 mathematical reasoning benchmarks. The “Toks. saving” column indicates the reduction in stored KV cache tokens compared to the generation length of the GRPO-Dense during RL training. Our key findings are as follows:

Sparse-RL achieves comparable performance to dense upper bounds. Under the constraint of a fixed retained token budget, GRPO with naive sparse rollout performs poorly overall, while GRPO + Sparse-RL maintains performance comparable to the Dense GRPO baseline and even outperforms it in certain scenarios. On Qwen2.5-1.5B, Sparse-RL with R-KV achieves an average score of 36.2, outperforming GRPO-Dense (35.4) by 2.3%, and attains the best results on 4 out of 7 benchmarks (GSM8K, Minerva, Olympiad, AMC23). On Qwen2.5-3B, Sparse-RL performs on par with

²huggingface.co/datasets/AI-MO/aimo-validation-aimo

³huggingface.co/datasets/AI-MO/aimo-validation-aimc

the dense baseline (42.8 vs. 43.4 on average). On Qwen2.5-7B, although a small performance gap remains (51.4 vs. 53.1 on average), Sparse-RL retains 96.8% of the dense model’s performance, indicating that the method scales effectively to larger models with more complex reasoning patterns.

Substantial reduction in memory overhead. As indicated by the “Toks. saving” metric, we achieve token savings of 35.1%, 53.3%, 42.0%, and 39.4% for the Llama3.2-1B-Instruct, Qwen2.5-1.5B, 3B, and 7B models, respectively. Notably, on the 1.5B model, GRPO + Sparse-RL outperformed the dense GRPO method while discarding over half of the contextual history (53.3% saving). This highlights the potential redundancy in standard CoT paths, showing that Sparse-RL enables the model to perform correct deductions using a much more compact information representation.

Robustness across models and compression algorithms. Consistent performance trends on both Qwen2.5 and Llama backbones indicate that our framework is robust across model architectures. Comparing two compression variants, Sparse-RL with R-KV and Sparse-RL with SnapKV, we observe highly similar results. For example, on Qwen2.5-3B, their average scores are nearly identical (42.8 vs. 42.6 on average). This indicates that Sparse-RL is largely compression-agnostic: as long as the compression method preserves the essential context, our off-policy correction mechanism can effectively stabilize training. Notably, R-KV shows a slight advantage on more challenging tasks such as Olympiad (27.7 vs. 25.9 on 3B, 36.6 vs. 32.9 on 7B), validating its design benefits in handling reasoning redundancy.

5.3 Training Dynamics

To further analyze the training stability and potential of Sparse-RL, we visualize the training dynamics of GRPO-Dense and GRPO + Sparse-RL (with R-KV) in Figure 2. Our analysis is as follows:

Outcome Rewards. The average reward trajectory of Sparse-RL is slightly lower than that of the dense baseline but remains within an acceptable margin, which is a direct consequence of the information loss inherent to compressed rollout generation. Despite the approximation errors introduced by KV compression, the reward curves indicate that Sparse-RL can still perform stable and effective policy optimization. While a persistent yet minor performance gap remains compared

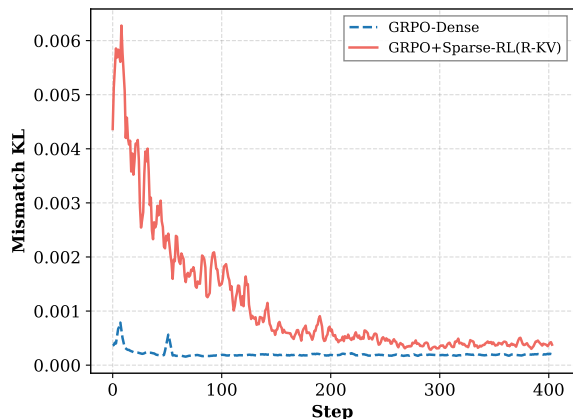


Figure 3: Comparison of the Mismatch KL between GRPO-Dense and GRPO + Sparse-RL (with R-KV) on Qwen2.5-7B.

to the GRPO-Dense baseline—which represents a theoretical upper bound—Sparse-RL successfully avoids the catastrophic training collapse commonly observed in naive sparse training settings.

Adaptation to Sparse Memory. The overall trend in response length for Sparse-RL (R-KV) and GRPO-Dense is to increase initially and then decrease. In the early stages of training, Sparse-RL (R-KV) has longer response length, possibly because the model generated long anomalous sequences (e.g., infinite repetitions) due to information loss. Crucially, as training progresses, the response length rapidly regresses to align with the Dense baseline. This convergence validates the corrective power of our RL framework, demonstrating that the model learns to perform token-efficient reasoning within the strict KV budget constraints.

Stability of Exploration. The GRPO-Dense baseline exhibits a steep drop in entropy during the initial phase, while Sparse-RL demonstrates a more gradual decay curve. Notably, Sparse-RL consistently maintains higher entropy levels between steps 50 and 300. We attribute this phenomenon to the implicit regularization effect of compression noise, which prevents the policy from prematurely overconfidently committing to shortcuts, thereby promoting exploration of the inference space. The entropy of Sparse-RL ultimately converges to the same level as the dense baseline, confirming that our method successfully achieves policy stability and avoids the divergence issues commonly encountered in sparse training.

Degree of Policy Mismatch. We quantify policy mismatch via the KL divergence between roll-

Model	GSM8K	MATH500	Gaokao	Minerva	Olympiad
Qwen2.5-1.5B					
GRPO (Dense)	73.4	50.8	37.7	12.5	14.2
↔ + Sparse-RL (R-KV)	74.6	54.0	42.9	15.4	18.1
Qwen2.5-3B					
GRPO (Dense)	85.1	54.2	44.2	16.5	17.3
↔ + Sparse-RL (R-KV)	84.1	61.8	49.1	22.1	23.3
Qwen2.5-7B					
GRPO (Dense)	91.2	71.6	55.3	33.8	33.0
↔ + Sparse-RL (R-KV)	91.4	71.8	59.0	30.9	33.5

Table 2: The performance of models trained by GRPO + Sparse-RL (with R-KV) and GRPO-Dense in the sparse inference scenario by R-KV compression.

out and training policies. As shown in Figure 3, Sparse-RL initially exhibits a higher KL magnitude 10^{-3} than the dense baseline 10^{-4} , reflecting the structural bias from KV compression. This discrepancy steadily decreases and stabilizes after approximately 200 steps. This convergence demonstrates that the dense learner successfully adapts to the compression logic, effectively bridging the distribution gap to learn consistent reasoning patterns from sparse trajectories.

Sampling Efficiency and Stability. To provide a more comprehensive view of our rejection sampling mechanism, we monitor the rejection ratio (the fraction of trajectories vetoed by the sparsity-aware filter) and policy gradient clipping ratio on Qwen2.5-3B throughout GRPO + Sparse-RL (R-KV) training. The average rejection rate is 0.07, indicating that the majority of generated trajectories satisfy the sparsity consistency constraint. The average clip ratio remains at 0.0005. The detailed dynamics are provided in Appendix C.

5.4 Bonus: Superiority in Sparse Inference

Beyond the standard evaluation, we investigate the models’ adaptability to downstream sparse inference scenarios. In this experiment, we enforce the exact same KV compression configuration used during Sparse-RL training (i.e., R-KV with a budget of 512 tokens) for models trained by the GRPO-Dense and our Sparse-RL (with R-KV). As shown in Table 2, Sparse-RL outperform their dense counterparts on the majority of benchmarks across varying scales. For instance, on Qwen2.5-3B, Sparse-RL surpasses the dense baseline by 7.6% on MATH500 under the sparse setting. The overall superiority highlights a critical insight: while GRPO-Dense suffers from a distribution shift when forcing sparsity at inference time, Sparse-

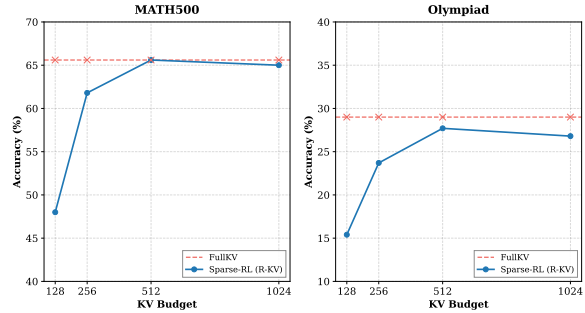


Figure 4: Ablation study on KV cache budget size. Performance of Sparse-RL (with R-KV) on Qwen2.5-3B across different budget levels on MATH500 and Olympiad-Bench. The red dashed line represents the dense rollouts baseline (FullKV).

RL successfully internalizes the compression logic during training. This *Sparsity-Aware Training* effectively aligns the training dynamics with the inference constraints, yielding significantly robust performance in memory-constrained applications.

5.5 Ablation Study

Size of KV Cache Budget. We evaluate the effect of different token budget sizes by varying the retained tokens during rollout (128, 256, 512, 1024, and FullKV) for Qwen2.5-3B trained with GRPO + Sparse-RL (using R-KV). As shown in Figure 4, performance degrades under extremely small budgets (128 tokens), indicating insufficient context for reliable reasoning. As the budget increases to 256 and 512 tokens, accuracy improves rapidly on both benchmarks. We note that for the models and benchmarks considered in this paper, a KV budget of 512 tokens is sufficient to achieve performance comparable to FullKV. For more challenging tasks or models with longer CoT generation, the optimal budget may need to be adjusted accordingly.

6 Conclusion

This paper investigates the critical memory bottleneck issue caused by key-value (KV) cache storage for rollout stage in reinforcement learning. We identify the policy mismatch between sparse sampler and dense learner as the root cause of training crashes under sparse rollout pattern. To address this, we propose Sparse-RL, a memory-efficient framework that enables sparse RL rollout. We integrate Sparsity-Aware Rejection Sampling and Importance-based Reweighting to ensure stable training. Experiments validate Sparse-RL’s effectiveness, achieving performance on par with

dense baselines under strict KV budget constraints while demonstrating robust resilience across varying model scales and compression methods.

Limitations

Task Generalization. While Sparse-RL demonstrates robust performance on reasoning-intensive tasks (e.g., mathematics) with verifiable binary rewards, its applicability to open-ended generation tasks—such as creative writing or general instruction following—remains under-explored. In these scenarios, the definition of anomalous tokens is more ambiguous, and the attention patterns may differ from the redundancy often found in Chain-of-Thought reasoning. Future work is needed to verify whether the distributional correction mechanisms of Sparse-RL generalize effectively to these broader domains.

Sampling Efficiency Trade-off. Sparse-RL relies on a strict rejection sampling mechanism to filter out trajectories corrupted by compression-induced information loss. Although this ensures training stability, it inherently introduces a trade-off in sampling efficiency. In cases where the compression budget is extremely aggressive or the policy mismatch is severe, the rejection rate may increase, leading to wasted computational resources on discarded trajectories. Optimizing this trade-off, perhaps through token-level correction during generation rather than sequence-level rejection, remains a direction for future improvement.

Acknowledgments

This work is supported by the National Key Research & Development Plan (2023YFF0725100) and the National Natural Science Foundation of China (92570121, 62322214, U23A20299, U24B20144).

References

Zefan Cai, Wen Xiao, Hanshi Sun, Cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, and 1 others. 2025. R-kv: Redundancy-aware kv cache compression for training-free reasoning models acceleration. *arXiv preprint arXiv:2505.24133*.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and 1 others. 2024. Pyramidkv: Dynamic kv cache compression based on

pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.

Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. 2025. Towards reasoning era: A survey of long chain-of-thought for reasoning large language models. *arXiv preprint arXiv:2503.09567*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv e-prints*, pages arXiv-2407.

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

Wei Fu, Jiakuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, Tongkai Yang, Binhang Yuan, and Yi Wu. 2025. [Areal: A large-scale asynchronous reinforcement learning system for language reasoning](#). *Preprint*, arXiv:2505.24298.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2023. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shitong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, and 1 others. 2024. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3828–3850.

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

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. 2024. Kvquant: Towards 10

- million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303.
- Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.
- Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, and 1 others. 2022. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857.
- 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.
- Jiacai Liu, Yingru Li, Yuqian Fu, Jiawei Wang, Qian Liu, and Yu Shen. 2025. [When speed kills stability: Demystifying rl collapse from the inference-training mismatch](#).
- Jiawei Liu and Lingming Zhang. 2025. Code-r1: Reproducing rl for code with reliable rewards. *arXiv preprint arXiv:2503.18470*, 3.
- Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyriilidis, and Anshumali Shrivastava. 2023. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364.
- Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.
- Ruoyu Qin, Weiran He, Weixiao Huang, Yangkun Zhang, Yikai Zhao, Bo Pang, Xinran Xu, Yingdi Shan, Yongwei Wu, and Mingxing Zhang. 2025. Seer: Online context learning for fast synchronous llm reinforcement learning.
- 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, Jiayi Yang, Jingren Zhou, and 25 others. 2025. [Qwen2.5 technical report](#). *Preprint, arXiv:2412.15115*.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Zelei Shao, Vikranth Srivatsa, Sanjana Srivastava, Qingyang Wu, Alpav Ariyak, Xiaoxia Wu, Ameen Patel, Jue Wang, Percy Liang, Tri Dao, and 1 others. 2025. Beat the long tail: Distribution-aware speculative decoding for rl training. *arXiv preprint arXiv:2511.13841*.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.
- Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. 2024. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. *arXiv preprint arXiv:2410.21465*.
- Qwen Team. 2025. Qwq-32b: Embracing the power of reinforcement learning.
- Siqi Wang, Hailong Yang, Junjie Zhu, Xuezhu Wang, Yufan Xu, and Depei Qian. 2025. Rlhfspec: Breaking the efficiency bottleneck in rlhf training via adaptive drafting. *arXiv preprint arXiv:2512.04752*.
- Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.
- Feng Yao, Liyuan Liu, Dinghuai Zhang, Chengyu Dong, Jingbo Shang, and Jianfeng Gao. 2025. [Your efficient rl framework secretly brings you off-policy rl training](#).
- Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. 2025. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *arXiv preprint arXiv:2503.18892*.
- Hanchen Zhang, Xiao Liu, Bowen Lv, Xueqiao Sun, Bohao Jing, Iat Long Iong, Zhenyu Hou, Zehan Qi, Hanyu Lai, Yifan Xu, and 1 others. 2025a. Agentrl: Scaling agentic reinforcement learning with a multi-turn, multi-task framework. *arXiv preprint arXiv:2510.04206*.

- Xiaotian Zhang, Chunyang Li, Yi Zong, Zhengyu Ying, Liang He, and Xipeng Qiu. 2023a. Evaluating the performance of large language models on gaokao benchmark. *arXiv preprint arXiv:2305.12474*.
- Xiaoyun Zhang, Jingqing Ruan, Xing Ma, Yawen Zhu, Haodong Zhao, Hao Li, Jiansong Chen, Ke Zeng, and Xunliang Cai. 2025b. When to continue thinking: Adaptive thinking mode switching for efficient reasoning. *arXiv preprint arXiv:2505.15400*.
- Xiaoyun Zhang, Xiaojian Yuan, Di Huang, Wang You, Chen Hu, Jingqing Ruan, Kejiang Chen, and Xing Hu. 2025c. Rediscovering entropy regularization: Adaptive coefficient unlocks its potential for llm reinforcement learning. *arXiv preprint arXiv:2510.10959*.
- Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. 2024. Cam: Cache merging for memory-efficient llms inference. In *Forty-first international conference on machine learning*.
- Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuan-dong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023b. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.
- Yilong Zhao, Jiaming Tang, Kan Zhu, Zihao Ye, Chih-Chih Chang, Chaofan Lin, Jongseok Park, Guangxuan Xiao, Mohamed S Abdelfattah, Mingyu Gao, and 1 others. 2025. Accelerating large-scale reasoning model inference with sparse self-speculative decoding. *arXiv preprint arXiv:2512.01278*.
- Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, and 1 others. 2025. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*.
- Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. 2025. slime: An llm post-training framework for rl scaling. <https://github.com/THUDDM/slime>. GitHub repository. Corresponding author: Xin Lv.

A Compression Configuration

The hyperparameters for R-KV and SnapKV include: (1) B_{budget} , the budget size of cache to store retained KV tokens; (2) B_{buffer} , the size of a buffer for newly generated text tokens. After the model generates each fixed-length text segment in the buffer, the KV cache compression is executed. (3) α , the last few tokens that are always retained in the cache as observation tokens. (4) λ (only in R-KV), controls the trade-off between prioritizing important tokens and reducing redundant tokens. In our experiments, $B_{\text{budget}} = 512$, $B_{\text{buffer}} = 128$, $\alpha = 8$, $\lambda = 0.1$, with the latter three parameters following the settings in the R-KV paper (Cai et al., 2025).

B Benchmark Statistics

The detailed statistical information of the 7 mathematical benchmarks is shown in Table 3.

Benchmark	Description	Size
GSM8K	High quality linguistically diverse grade school math word problems.	1319
MATH500	A subset from the MATH benchmark that OpenAI created in their <i>Let's Verify Step by Step</i> paper.	500
Minerva	A curated subset of Minerva targeting advanced STEM reasoning from undergraduate math and science exams.	272
Gaokao	Math questions from Chinese college entrance examinations.	385
Olympiad	Olympiad-level mathematics and physics competitions.	675
AIME24	Problems from the 2024 American Invitational Mathematics Examination.	30
AMC23	Problems from the 2023 American Mathematics Competition.	40

Table 3: Description of benchmark

C Dynamics of Rejection Rate and Clip Ratio

We monitor the rejection rate and the policy gradient clipping ratio, using Qwen2.5-3B trained by GRPO + Sparse-RL (with R-KV).

Figures 5 and 6 illustrate the detailed training dynamics on Qwen2.5-3B. The rejection ratio (Figure 5) fluctuates between roughly 0.05 and 0.11 throughout the training process, with an average of 0.07. This indicates that the vast majority of trajectories generated by the sparse policy satisfy the

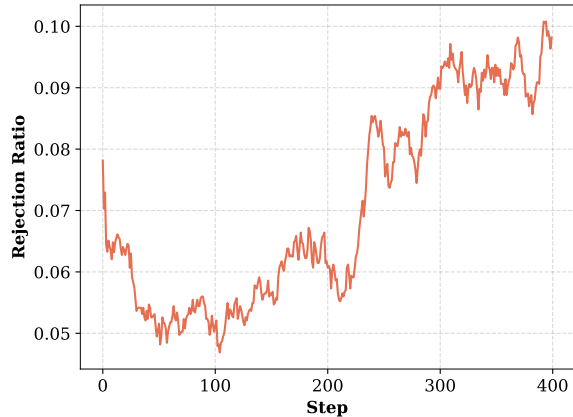


Figure 5: Dynamics of rejection rate on Qwen2.5-3B trained by GRPO + Sparse-RL (with R-KV).

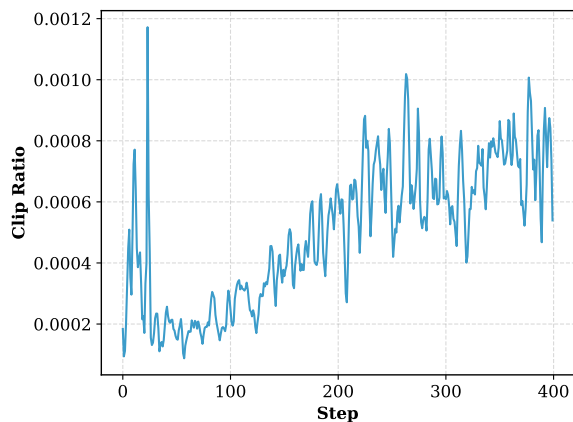


Figure 6: Dynamics of policy gradient clip ratio on Qwen2.5-3B trained by GRPO + Sparse-RL (with R-KV).

consistency constraints required by our rejection sampling mechanism. Meanwhile, the clip ratio (Figure 6) remains negligible in the order of 10^{-4} , confirming that our Importance-Based Reweighting successfully corrects off-policy bias, keeping policy updates well-contained within a stable trust region throughout training.

D Preliminaries of GRPO

Group Relative Policy Optimization (GRPO), introduced by (Shao et al., 2024), is a variant of the Proximal Policy Optimization (PPO) algorithm specifically designed for LLMs. The key innovation of GRPO is the elimination of the value model (critic), which significantly reduces computational overhead.

GRPO estimates the advantage of a response by comparing its reward against the relative performance within a group of samples generated from

the same prompt. For a given input prompt x , the model generates G responses $\{o_1, o_2, \dots, o_G\}$. Each response o_i is assigned a reward r_i (through a rule-based verifier or a reward model). The advantage \hat{A}_i for each response o_i is calculated as:

$$\hat{A}_i = \frac{r_i - \text{mean}(r_1, \dots, r_G)}{\text{std}(r_1, \dots, r_G)} \quad (10)$$

Based on this, GRPO adopts a clipped surrogate loss similar to PPO:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{\substack{x \sim \mathcal{D} \\ o_i \sim \pi_{\theta_{\text{old}}}}} \left[\frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \left(\min(w_{i,t}(\theta) \hat{A}_i, \text{clip}(w_{i,t}(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_i) \right) \right] \quad (11)$$

where $w_{i,t}(\theta) = \frac{\pi_{\theta}(o_{i,t}|x, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|x, o_{i,<t})}$ is the importance sampling weight (policy ratio), ϵ is the clipping threshold.

E Derivation of Sparse-RL Objective Function

According to section 4.3, the objective is:

$$\mathcal{J}_{\text{Sparse-RL}}(\theta) = \mathbb{E}_{\substack{x \sim \mathcal{D} \\ o_i \sim \pi_{\theta_{\text{sparse}}}}} \left[\frac{1}{G} \sum_{i=1}^G M^{\text{RS}}(o_i) \cdot \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \xi_{i,t} \cdot \min(w_{i,t}(\theta) \hat{A}_i, \text{clip}(w_{i,t}(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_i) \right] \quad (12)$$

The gradient derivation of Eq.12 is (clipping is omitted for brevity):

$$\begin{aligned} \nabla_{\theta} \mathcal{J}_{\text{CaRPO}}(\theta) &= \nabla_{\theta} \mathbb{E}_{\substack{x \sim \mathcal{D} \\ o_i \sim \pi_{\theta_{\text{sparse}}}}} \left[\frac{1}{G} \sum_{i=1}^G \frac{M^{\text{RS}}(o_i)}{|o_i|} \sum_{t=1}^{|o_i|} \xi_{i,t} \cdot w_{i,t}(\theta) \hat{A}_i \right] \\ &= \mathbb{E}_{\substack{x \sim \mathcal{D} \\ o_i \sim \pi_{\theta_{\text{sparse}}}}} \left[\frac{1}{G} \sum_{i=1}^G \frac{M^{\text{RS}}(o_i)}{|o_i|} \sum_{t=1}^{|o_i|} \xi_{i,t} \cdot \nabla_{\theta} w_{i,t}(\theta) \hat{A}_i \right] \end{aligned} \quad (13)$$

By applying the chain rule to the policy ratio $w_{i,t}(\theta) = \frac{\pi_{\theta}(o_{i,t}|x, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|x, o_{i,<t})}$, the gradient term can be expanded:

$$\nabla_{\theta} w_{i,t}(\theta) = \frac{\pi_{\theta}(o_{i,t}|x, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|x, o_{i,<t})} \nabla_{\theta} \log \pi_{\theta}(o_{i,t}|x, o_{i,<t}) \quad (14)$$

Substituting this back into Eq. 13, we obtain the final gradient update:

$$\nabla_{\theta} \mathcal{J}_{\text{Sparse-RL}} = \mathbb{E}_{\substack{x \sim \mathcal{D} \\ o_i \sim \pi_{\theta_{\text{sparse}}}}} \left[\frac{1}{G} \sum_{i=1}^G M^{\text{RS}}(o_i) \cdot \frac{\hat{A}_i}{|o_i|} \cdot \sum_{t=1}^{|o_i|} (\xi_{i,t} \cdot w_{i,t}(\theta) \nabla_{\theta} \log \pi_{\theta}(o_{i,t}|x, o_{i,<t})) \right] \quad (15)$$

Because we have:

$$\begin{aligned} \xi_{i,t} \cdot w_{i,t}(\theta) &= \frac{\pi_{\theta_{\text{old}}}(o_{i,t}|x, o_{i,<t})}{\pi_{\theta_{\text{sparse}}}(o_{i,t}|x, o_{i,<t})} \cdot \frac{\pi_{\theta}(o_{i,t}|x, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|x, o_{i,<t})} \\ &= \frac{\pi_{\theta}(o_{i,t}|x, o_{i,<t})}{\pi_{\theta_{\text{sparse}}}(o_{i,t}|x, o_{i,<t})} \end{aligned} \quad (16)$$

Substituting Eq. 16 into Eq. 15, the gradient becomes:

$$\nabla_{\theta} \mathcal{J}_{\text{Sparse-RL}} = \mathbb{E}_{\substack{x \sim \mathcal{D} \\ o_i \sim \pi_{\theta_{\text{sparse}}}}} \left[\frac{1}{G} \sum_{i=1}^G M^{\text{RS}}(o_i) \cdot \frac{\hat{A}_i}{|o_i|} \cdot \sum_{t=1}^{|o_i|} \left(\frac{\pi_{\theta}(o_{i,t}|x, o_{i,<t})}{\pi_{\theta_{\text{sparse}}}(o_{i,t}|x, o_{i,<t})} \nabla_{\theta} \log \pi_{\theta}(o_{i,t}|x, o_{i,<t}) \right) \right] \quad (17)$$

F Decoding Efficiency and Latency Analysis

Sequence Length	Dense(ms)	SnapKV(ms)	Speedup
8k	21.36	21.61	$\sim 1.0 \times$
16k	28.70	21.58	$\sim 1.33 \times$
32k	43.99	21.57	$\sim 2.04 \times$

Table 4: Operator-level decoding latency comparison between dense baseline and SnapKV on Qwen3-8B across different sequence lengths.

To better understand the computational impact of KV cache compression, we conduct an operator-level benchmark on autoregressive decoding latency using Qwen3-8B. We compare a standard dense decoding baseline with sparse decoding enabled by SnapKV, one of the compression methods used in our framework.

Specifically, we measure the average per-step decoding latency under different sequence lengths (8k, 16k, and 32k tokens). The results are summarized in Table 4.

We observe that the baseline decoding latency increases approximately linearly with sequence length, reflecting the growing cost of attention over the full KV cache. In contrast, SnapKV maintains a near-constant decoding latency across all sequence lengths, effectively decoupling computation from context length.

In addition to latency improvements, KV compression reduces KV cache storage by 35–53% in our main experiments. This reduction directly translates into improved memory efficiency, enabling larger rollout batch sizes and supporting longer context generation without triggering out-of-memory (OOM) errors during RL training.

G Example of Anomalous Sequence

Prompt & Dense Rollout (Correct)

Prompt:

The expression $2 \cdot 3 \cdot 4 \cdot 5 + 1$ is equal to 121, since multiplication is carried out before addition. However, we can obtain values other than 121 for this expression if we are allowed to change it by inserting parentheses. For example,

$$(2 \cdot (3 \cdot 4)) \cdot (5 + 1) = 144.$$

In total, how many values can be obtained?

Dense Rollout (Correct):

We can reason step by step as follows:

1. We have the expression $2 \cdot 3 \cdot 4 \cdot 5 + 1$.
2. We can insert parentheses to change the order of operations and obtain different values.
3. Let's consider each possible position for inserting parentheses:
 - (a) Between 2 and 3: $(2 \cdot 3) \cdot 4 \cdot 5 + 1$
 - (b) Between 3 and 4: $2 \cdot (3 \cdot 4) \cdot 5 + 1$
 - (c) Between 4 and 5: $2 \cdot 3 \cdot (4 \cdot 5) + 1$
 - (d) Between 5 and 1: $2 \cdot 3 \cdot 4 \cdot (5 + 1)$
4. We can calculate the value of each modified expression:
 - (a) $(2 \cdot 3) \cdot 4 \cdot 5 + 1 = 6 \cdot 4 \cdot 5 + 1 = 120 + 1 = 121$
 - (b) $2 \cdot (3 \cdot 4) \cdot 5 + 1 = 2 \cdot 12 \cdot 5 + 1 = 120 + 1 = 121$
 - (c) $2 \cdot 3 \cdot (4 \cdot 5) + 1 = 2 \cdot 3 \cdot 20 + 1 = 120 + 1 = 121$
 - (d) $2 \cdot 3 \cdot 4 \cdot (5 + 1) = 2 \cdot 3 \cdot 4 \cdot 6 = 144$
5. We have found four different values that can be obtained by inserting parentheses in the expression: 121, 121, 121, and 144.
6. Therefore, the total number of values that can be obtained from the expression $2 \cdot 3 \cdot 4 \cdot 5 + 1$ by inserting parentheses is 4. ✓

Sparse Rollout (Anomalous)

To determine how many distinct values can be obtained from the expression $2 \cdot 3 \cdot 4 \cdot 5 + 1$ by inserting parentheses, we need to consider all possible ways to distribute multiplication operations across the sequence of numbers and the addition. Let's break this down step by step.

Step 1: List Out All Possible Parenthesizations

We start by listing out all potential combinations of inserting parentheses into the expression $2 \cdot 3 \cdot 4 \cdot 5 + 1$. We will consider different groupings of the multiplications, but not changing the overall structure of the additions or subtractions, as only the grouping of the multiplications is allowed.

Here are the possible ways:

1. $(2 \cdot 3) \cdot (4 \cdot 5) + 1$
2. $2 \cdot (3 \cdot (4 \cdot 5)) + 1$
3. $2 \cdot (3 \cdot (4 \cdot (5 + 1)))$
4. $((2 \cdot 3) \cdot 4) \cdot 5 + 1$
5. $2 \cdot ((3 \cdot 4) \cdot 5) + 1$
6. $2 \cdot (3 \cdot (4 \cdot (5 + 1)))$
7. $2 \cdot 3 \cdot 4 \cdot (5 + 1) + 1$
8. $(2 \cdot 3 \cdot (4 \cdot 5)) + 1$
9. $2 \cdot ((3 \cdot 4) \cdot (5 + 1)) + 1$

Step 2: Calculate Each Parenthesized Expression

Now, let's calculate each of these expressions individually to find out their respective values:

1. $(2 \cdot 3) \cdot (4 \cdot 5) + 1 = 6 \cdot 20 + 1 = 120 + 1 = 121$
2. $2 \cdot (3 \cdot (4 \cdot 5)) + 1 = 2 \cdot (3 \cdot 20) + 1 = 2 \cdot 60 + 1 = 120 + 1 = 121$
3. $2 \cdot (3 \cdot (4 \cdot (5 + 1))) = 2 \cdot (3 \cdot (4 \cdot 6)) + 1 = 2 \cdot (3 \cdot 24) + 1 = 2 \cdot 72 + 1 = 144 + 1 = 145$
4. $((2 \cdot 3) \cdot 4) \cdot 5 + 1 = (6 \cdot 4) \cdot 5 + 1 = 24 \cdot 5 + 1 = 120 + 1 = 121$
5. $2 \cdot ((3 \cdot 4) \cdot 5) + 1 = 2 \cdot (12 \cdot 5) + 1 = 2 \cdot 60 + 1 = 120 + 1 = 121$
6. $2 \cdot (3 \cdot (4 \cdot (5 + 1))) = 2 \cdot (3 \cdot 24) + 1 = 2 \cdot 72 + 1 = 144 + 1 = 145$

$$141 + 1 = 141 + 1 = 141 + 1 = 141 + 1 \dots$$

[Infinite repetition]

Figure 7: The information loss caused by KV cache compression may generate anomalous sequences (e.g., infinite repetition) during rollout phase. We provide a concrete example sampled during the training of Qwen2.5-3B with R-KV compression (budget=512).