S^{3}R: FAST AND FAITHFUL RL ROLLOUTS VIA SELF-SPARSE SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Fast rollout generation for long-context reinforcement learning with verifiable rewards (RLVR) requires reducing KV-cache traffic without changing the response groups used for policy optimization. In our profiled RLVR workloads, attention accounts for 72–76% of autoregressive decoding kernel time. Sparse attention reduces KV-cache reads, but sampling directly from a sparse policy changes this distribution. For group-relative optimization, we show that even exact response-wise reweighting does not generally recover the expected dense-policy update because the advantage of each response depends on the rewards of other group members. We propose Self-Sparse Speculative Rollouts (S^{3}R). It uses sparse attention to propose tokens, then generates responses through full-attention verification and exact speculative sampling. Drafting and verification share the same policy parameter, eliminating the need to train or update a separate drafter. The resulting response groups follow the dense-policy distribution. We characterize the conditions under which S^{3}R achieves a speedup and analyze the costs of drafting and verification. We find that dense verification still requires the complete historical KV cache. A KV-aware scheduler therefore reduces repeated offloading and restoration of the cache while maintaining rollout parallelism. In a matched ablation with an aggressive sparse drafting budget, S^{3}R maintains stable training while direct sparse sampling with training-side correction becomes unstable. For long-context mathematical reasoning with Qwen3, S^{3}R achieves $1.35 \sim 1.39\times$ the end-to-end training throughput of autoregressive rollouts, with comparable task performance. Our code is available at `https://anonymous.4open.science/r/S3R-7666`

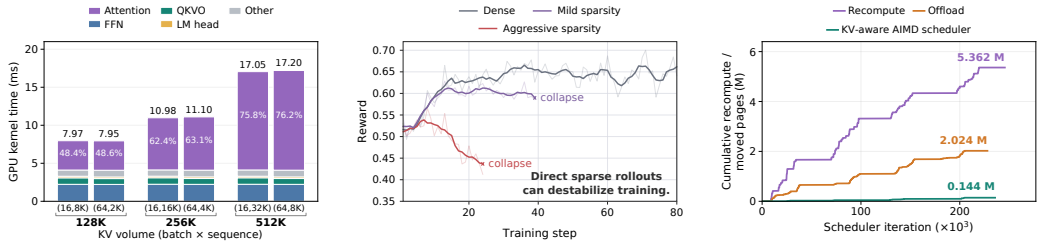
1 INTRODUCTION

Rollout generation is a primary bottleneck in reinforcement learning with verifiable rewards (RLVR) for large language models (LLMs) (OpenAI, 2024; DeepSeek-AI, 2025). Algorithms such as Group Relative Policy Optimization (GRPO) samples groups of responses and compute group-relative advantages (Shao et al., 2024). Generating long responses through autoregressive (AR) decoding makes rollouts a substantial component of the overall training pipeline (Kim et al., 2026; Wang et al., 2025b; Zhou et al., 2025), motivating methods of reducing rollout cost for efficient RLVR.

In our profiled workloads, attention accounts for much of this cost: each decoding step read the KV cache associated with an increasingly long context (Figure 1a). Training-free KV compression and sparse attention can reduce these reads by restricting context (Li et al., 2024; Tang et al., 2024; Cai et al., 2025). When used directly for sampling, sparse attention alters the behavior policy and can destabilize training (Luo et al., 2026; Zhou et al., 2026b) (Figure 1b). Training-side filtering and importance reweighting can mitigate this mismatch (Luo et al., 2026). In GRPO, however, each response's advantage depends on the rewards of other group members. We show that even exact response-wise reweighting need not recover the expected dense policy update. We therefore seek to accelerate rollout generation through sparse computation while preserving the dense policy's response-group distribution.

Speculative decoding (SD) preserves the target distribution through exact sampling, using a cheaper drafter to propose candidate tokens and the target (i.e., the policy itself) to verify them in parallel (Leviathan et al., 2023; Chen et al., 2023; Zhou et al., 2024). In practice, the speedup from SD

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) Attention dominates AR. (b) Sparse-rollout collapse. (c) KV-aware scheduling.

Figure 1: (a) Attention accounts for 72–76% of AR-decoding kernel time as KV volume grows (Full results in Appendix F.2.1). (b) Both mild and aggressive sparsity exhibit training collapse without training-side mismatch correction. (c) Cumulative KV pages recomputed or transferred; our scheduler substantially reduces repeated KV reconstruction and transfer.

depends on the availability of a suitable drafter with sufficient acceptance. Such a drafter may not be readily available for fine-tuned target models and may require additional training and deployment effort. Moreover, repeated policy updates in RLVR can cause a fixed drafter to become stale, further reducing the acceptance rate. Auxiliary proposers may produce accurate proposals, but maintaining alignment between the proposer and the evolving policy introduces optimization and systems overhead (Zhang et al., 2025; Chen et al., 2025b; Hu et al., 2026). History retrieval methods avoid an auxiliary model but rely on past trajectories covering continuations under the evolving policy (Liu et al., 2025; Shao et al., 2025; He et al., 2025). For both approach, decoding progress must amortize drafting and verification costs across changing workloads (Kim et al., 2026). These considerations motivate using sparsification of the behavior policy to generate low-cost proposals, while retaining dense verification to sample faithfully from the target distribution.

To this end, we present **S**elf-**S**parse **S**peculative **R**ollouts (S^3R), a framework for fast and faithful RLVR rollouts. We first formalize and analyze the mismatch induced by sparsity at the response-group level and show why exact response-level reweighting does not necessarily recover the expected group update under the dense policy. Based on this analysis, S^3R confines sparsity to proposal generation and combines full-attention verification with exact speculative sampling to preserve the dense response-group distribution. Drafting and verification share the same policy weights, keeping the drafter synchronized without separate drafter fine-tuning. Next, to accommodate the changing RLVR workloads, we characterize how S^3R amortizes drafting and verification costs in the runtime. During rollout generation, dense verification retains the complete historical KV cache. Under GPU memory pressure, repeated KV offloading and restoration can cause cache thrashing. We introduce KV-aware scheduling to limit this thrashing while sustaining rollout parallelism (Figure 1c). Cross-step state management supports policy synchronization and inference restoration after each actor update. Finally, experiments on long-context mathematical reasoning demonstrate stable training under aggressive sparse drafting and higher end-to-end training throughput than AR, with comparable task performance. Our contributions are threefold:

- **Group-level mismatch analysis.** We show that exact response-wise reweighting need not recover the dense policy's expected update, and provide formulation and design principles for RL rollout.
- **Faithful sparse self-speculative rollouts.** We develop S^3R by integrating sparse self-drafting with exact sampling, preserving the dense response-group distribution without separate drafter training. We characterize its speedup conditions and provide runtime support for efficient RLVR.
- **Practical long-context acceleration.** On mathematical RLVR workloads, S^3R increases end-to-end training throughput over AR by 1.39× on Qwen3-1.7B and 1.35× on Qwen3-8B, while maintaining stable training and comparable task performance.

2 Related Work

2.1 Efficient Reinforcement Learning for LLMs

RL post-training methods improve the efficiency and stability of policy optimization (Schulman et al., 2017; Shao et al., 2024; DeepSeek-AI, 2025; Zheng et al., 2025; Li et al., 2025a). System work further improves GPU utilization through flexible orchestration, asynchronous overlap, and tail-aware scheduling (Sheng et al., 2025; Fu et al., 2025; Qin et al., 2025; Zhang et al., 2026a).

Nevertheless, rollout generation remains sequential across token positions, and long responses increase both decoding latency and KV-cache pressure (Kim et al., 2026; Kwon et al., 2023). These systems improve the orchestration of RL training and are complementary to reducing the cost of sampling each rollout.

2.2 Sparse and Other Approximate RL Rollouts

Approximate rollouts (e.g. Sparsification) reduce generation cost but sample from a policy that differs from the dense policy rollout. Sparse-attention approaches alleviate this gap through filtering, importance weighting, or approximation schedules constrained by distributional statistics (Luo et al., 2026; Zhou et al., 2026b). Low-precision rollout methods instead use mismatch-aware optimization or align training and inference numerics (Li et al., 2026; Ling Team & Inclusion AI, 2025; Zhou et al., 2026a; Gu et al., 2026; Qi et al., 2025). Other approaches reduce generation cost through early-exit inference or proxy rollout models (Elhoushi et al., 2024; Chen et al., 2026c). These mechanisms control or compensate for approximation in rollout generation. S^{3}R instead uses self-sparse proposals and full-attention verification before reward and advantage computation. It targets long-context KV traffic without changing the standard dense rollout-policy distribution.

2.3 Speculative Decoding for Inference and RL Rollouts

Speculative decoding amortizes autoregressive decoding by verifying multiple draft tokens in parallel and, under exact rejection sampling, preserves the target distribution (Leviathan et al., 2023; Chen et al., 2023). Methods mainly differ in their draft source. Retrieval-based methods reuse stored text or cached rollouts, avoiding drafter training but depending on recurrence between available proposals and new trajectories (He et al., 2024; Liu et al., 2025; Chang et al., 2026; Shao et al., 2025). Auxiliary proposers can be adapted online to changing policies or serving workloads, at the cost of maintaining and updating a separate network (Zhang et al., 2025; Chen et al., 2025b; Hu et al., 2026; Wang et al., 2026a). Target-induced methods instead derive the drafter from the current policy. They use layer skipping (Zhang et al., 2024; Xia et al., 2025; Chen et al., 2025a; Song et al., 2026), sparse attention (Zhao et al., 2025; Yue et al., 2026; Liu et al., 2026b), or quantization (Tiwari et al., 2025) to reduce drafting cost. S^{3}R uses sparse-attention self-drafting for group-based RLVR: it analyzes the limitations of response-wise correction, specifies conditions for preserving dense response groups, and supports repeated rollout–optimization transitions. Concurrent EfficientRollout (Kim et al., 2026) uses quantized self-drafting to reduce weight-loading costs. Both approaches avoid separate drafter training, but target different sources of rollout cost.

3 Problem Formulation and Design Principles

3.1 Sparsity-induced Mismatch in RLVR

In RLVR, policy-gradient updates depend on the behavior policy that generates responses to a prompt x. Let o^i denote the i-th complete response, o^i_t its token at position t, and $o^i_{<t}$ its preceding prefix. The learner assigns probability $\pi_\theta(o^i_t \mid x, o^i_{<t})$ to next token generation. The behavior policy is typically a frozen dense snapshot $\pi^{\text{dense}}_{\theta_{\text{old}}}$ of the learner. For each prompt, G responses sampled independently from this policy form a group $\mathcal{O} = (o^1, \ldots, o^G)$, whose rewards determine the group-relative advantages used in policy optimization.

To reduce rollout generation cost, we investigate sparsification of $\pi^{\text{dense}}_{\theta_{\text{old}}}$ as a rollout generation mechanism. Sparse attention restricts KV reads without changing the policy parameters, but changes the resulting token distribution. Running the frozen snapshot with a sparse budget $s \in (0, 1]$ defines a proposal policy $\pi^{\text{sparse}}_{\theta_{\text{old}},s}$. If sparse responses are used directly for training, $\pi^{\text{sparse}}_{\theta_{\text{old}},s}$ becomes the behavior policy. The corresponding token-level importance sampling (IS) ratio decomposes as:

$$\hat{w} = \underbrace{\frac{\pi_\theta(o^i_t \mid x, o^i_{<t})}{\pi^{\text{dense}}_{\theta_{\text{old}}}(o^i_t \mid x, o^i_{<t})}}_{w(\theta)\ \text{standard policy update}} \times \underbrace{\frac{\pi^{\text{dense}}_{\theta_{\text{old}}}(o^i_t \mid x, o^i_{<t})}{\pi^{\text{sparse}}_{\theta_{\text{old}},s}(o^i_t \mid x, o^i_{<t})}}_{w_s\ \text{sparsity-induced mismatch}}. \tag{1}$$

where $w(\theta)$ captures the standard discrepancy caused by policy updates, and w_s arises from sparsification. Our objective is to accelerate generation while preserving the faithfulness of the final

response groups by $\pi^{\text{dense}}_{\theta_{\text{old}}}$. Because GRPO advantages depend on the complete group, we next examine whether correcting each response separately is sufficient to recover the dense policy update.

3.2 Implications of Sparse Rollouts for GRPO Updates

Training-side methods mitigate the mismatch in Eq. 1 by reweighting or filtering sampled responses (Luo et al., 2026; Ling Team & Inclusion AI, 2025). To isolate the effect of group-relative advantage computation, we consider an idealized correction that uses exact response likelihood ratios rather than local token ratios (e.g., IS). This asks whether reweighting each response's own probability is sufficient when its update depends on the other group members. We denote $P(o^i \mid x) = \pi^{\text{dense}}_{\theta_{\text{old}}}(o^i \mid x)$ and $Q_s(o^i \mid x) = \pi^{\text{sparse}}_{\theta_{\text{old}},s}(o^i \mid x)$ as the probabilities of the complete response o^i under dense and sparse rollout, respectively.

Proposition 1 (Residual GRPO Update Bias under Response-Wise Reweighting)**.** *Fix the learner and rollout parameters. Multiplying each sparse response's otherwise unchanged update contribution by its exact response weight $W_i = P(o^i \mid x)/Q_s(o^i \mid x)$, does not, in general, recover the expected update under dense response groups.*

The weight W_i corrects for response i's own sampling mismatch, but its GRPO advantage still depends on the rewards of other responses drawn from Q_s. Thus, sparse sampling changes not only which response is observed, but also the reward comparisons that determine its group-relative advantage. Appendix A provides the supporting assumptions and proofs. To retain the baseline update without redesigning group-level correction, we place the sparsification in candidate generation and preserve faithfulness of the response groups to the dense policy.

3.3 Preserving Dense Response Groups via Exact Sampling

This placement requires a sampling procedure whose outputs follow taregt distribution, even when its candidates are not. We achieve this separation through exact speculative sampling. At each candidate position, write $p(v)$ and $q_s(v)$ for the target and drafted candidate probabilities at the same prefix. Standard speculative sampling (Leviathan et al., 2023; Chen et al., 2023) checks candidates in token order, accepting each with probability $\min\{1, p(v)/q_s(v)\}$ using fresh randomness. At the first rejection, the rejected candidate and its follow-up suffix are discarded, and a correction token is sampled from the residual distribution.

$$r_s(v) = \frac{[p(v) - q_s(v)]_+}{\sum_u [p(u) - q_s(u)]_+}, \tag{2}$$

where $[z]_+ = \max\{z, 0\}$. Acceptance and residual sampling together recover the dense conditional distribution at verified positions.

Proposition 2 (Dense response-group preservation)**.** *Fix the rollout snapshot θ_{old} and prompt x. Generate all G responses independently with sparse proposals q_s and exact speculative sampling targeting p, using the dense baseline's sampling rules. For any valid sparse budget s, the resulting response-group distribution $\Pr_{S^3R,s}(\mathcal{O} \mid x)$ matches the dense AR response-group distribution:*

$$\Pr_{S^3R,s}(\mathcal{O} \mid x) = \prod_{i=1}^{G} P(o^i \mid x), \qquad \mathcal{O} = (o^1, \ldots, o^G). \tag{3}$$

At fixed learner parameters, the training pipeline therefore produces the same distribution as dense AR rollouts. The learner uses the standard ratio $w(\theta)$ in Eq. 1, without an additional sparsity-induced correction. Appendix A gives the full conditions and proof. Under these conditions, the sparse budget changes decoding cost, but not the response-group distribution. It can therefore be selected for generation efficiency. We instantiate this separation using sparsification of the $\pi^{\text{dense}}_{\theta_{\text{old}}}$, avoiding a separately trained drafter. Section 4 specifies the drafting and dense verification mechanisms.

4 S^{3}R: Self-Sparse Speculative Rollouts for Reinforcement Learning

S^{3}R implements the separation through two coupled designs. (1) Self-drafting with sparse KV states to reduce latency without a separately trained proposer. (2) Full-attention verification supplies the

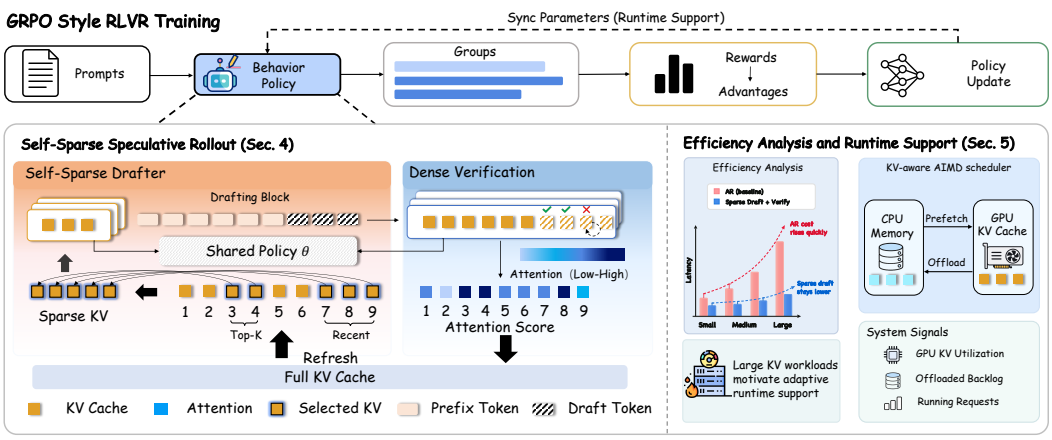


Figure 2: Overview of S^{3}R's rollout mechanism and runtime support. Sparse drafting reduces RL rollout cost, while dense verification preserves rollout fidelity with KV-aware runtime support.

probabilities for exact sampling and recomputes dense states for subsequent generation. Both designs share a snapshot (θ_{old}), synchronized after each actor update and held fixed until all requested responses are complete. Figure 2 illustrates these connections, and Algorithm 1 gives the rollout and policy-update loop.

Sparse Drafting from the Shared Policy. Dense prefill generates the first output token. The drafter uses the full KV cache for the first block. For each later block, it selects historical KV entries using attention scores from the previous verification pass. Let $\mathcal{C}_\ell$ denote the valid cached KV index set at layer ℓ when the sparse context is refreshed. The configured sparse budget ratio s specifies the target fraction of these indices to retain. For each KV head (head indices omitted), we retain recent cached positions $\mathcal{R}_\ell \subseteq \mathcal{C}_\ell$ around the verification boundary and select the remaining positions using the verifier's attention scores $\mathbf{a}_\ell$:

$$\mathcal{I}_\ell = \mathcal{R}_\ell \cup \text{TopK}_{\lceil s|\mathcal{C}_\ell|\rceil - |\mathcal{R}_\ell|}(\mathbf{a}_\ell). \tag{4}$$

Here $|\mathcal{R}_\ell| \leq \lceil s|\mathcal{C}_\ell|\rceil$, and Top-K selects the highest-scoring indices in $\mathcal{C}_\ell \setminus \mathcal{R}_\ell$. Each draft step applies attention to this view of the complete KV cache. Across k autoregressive steps, it holds the historical selection fixed and extends $\mathcal{I}_\ell$ with provisional KV entries for newly processed tokens. The drafter records each candidate and its full proposal distribution for the exact sampling step.

Full-Attention Verification and Context Refresh. The verifier evaluates the draft tokens in a single full-attention causal forward. Full attention alone is not sufficient if it reuses approximate historical states; verification therefore operates on dense historical KV and recomputes the proposed block through all layers. Then, we apply the exact sampling in Section 3.3. Verification replaces provisional draft states with recomputed dense KV entries, and removes the unused candidate suffix. The verifier's attention weights are averaged to obtain $\mathbf{a}_\ell$. After verification, we update $\mathcal{C}_\ell$ and $\mathcal{R}_\ell$ for the accepted prefix and refresh $\mathcal{I}_\ell$ using the verifier's scores in Eq. 4. Once all requested responses are complete, the trainer assembles them by prompt and group identity and computes rewards and group-relative advantages.

5 Efficiency Analysis and Runtime Support

Self-sparse drafting reduces historical KV reads but retains projections and feed-forward computation, and dense verification. Its benefit therefore depends on the cost of SD and on whether runtime state management preserves the resulting computational savings.

Latency Analysis. Consider a locally fixed active batch size B and cached context length S. Let T_p and T_q denote the dense and sparse per-step latencies, and T_V the verification latency. Each round drafts k tokens and outputs τ tokens on average per request (block efficiency), including a

correction or bonus token when applicable. For sequential drafting and verification, the estimated round latency T_{SD} and speedup η over AR are (Leviathan et al., 2023; Sadhukhan et al., 2025):

$$T_{\mathrm{SD}} = kT_q + T_V, \qquad \eta = \frac{T_p}{T_{\mathrm{SD}}/\tau}. \tag{5}$$

Acceleration therefore requires $\eta > 1 \iff \tau > (kT_q + T_V)/T_p$. The sparse budget therefore trades off drafting cost against accepted progress: higher acceptance is useful only when its benefit exceeds the additional drafting cost.

Figure 3 illustrates this trade-off using separately measured stage latencies with $k = 6$. At $(B, S) = (16, 4\mathrm{K})$, drafting and verification already exceed the $7T_p$ AR reference, even with full acceptance. With an aggregate KV volume $N_{\mathrm{KV}} = B \cdot S \approx 5.24 \times 10^5$ tokens, the corresponding break-even block efficiency falls to about 3.8 output tokens per round (Table 7). These serialized estimates exclude no-forward stalls and cross-request phase mixing; Appendix F.1 extends the analysis to mixed-stage execution.

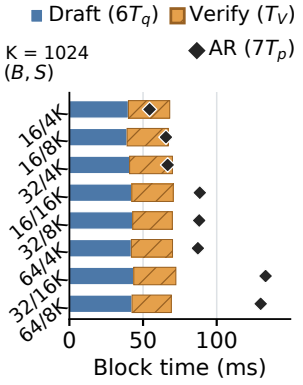


Figure 3: Trade-off costs on Qwen3-8B ($s = 0.05$, $k = 6$).

Runtime Support for RL Rollouts. Dense verification retains the full historical KV cache even when sparse drafting reduces reads. Under GPU memory pressure, repeated eviction and recovery can cause KV-cache thrashing (Chen et al., 2026b). CPU offloading avoids prefix recomputation, but repeated KV transfers can still consume the available latency margin.

A permanently low concurrency limit can reduce thrashing but sacrifices batching when capacity is available. We use a KV-aware additive increase and multiplicative decrease (AIMD) scheduler to adapt request admission. GPU KV utilization measures resident pressure, while the restoration backlog records outstanding demand from paused requests. Under low utilization, the concurrency limit increases additively; otherwise, high utilization or an excessive backlog triggers multiplicative backoff. Paused requests are restored before new requests are admitted, subject to the concurrency limit and available KV capacity.

Paused requests retain their token and proposal states. All requested responses complete under the same frozen rollout snapshot and retain their original group identities, maintaining the complete-group requirement of Proposition 2. Between RL steps, the runtime releases request and KV state for actor optimization, then synchronizes the updated policy and rebuilds inference state before the next rollout. Appendix B provides the scheduling rules and lifecycle details.

6 Experiments

6.1 Setup

Training Details. We initialize policies from Qwen3-1.7B and Qwen3-8B (Qwen Team, 2025) and train them on DAPO-MATH-17k (Yu et al., 2025). Optimization follows the DAPO variant of GRPO (Shao et al., 2024). Each batch contains 256 prompts with eight responses per prompt, sampled at temperature 1.0. The maximum response lengths are 16K and 24K tokens for the 1.7B and 8B models, respectively. All methods run in veRL (Sheng et al., 2025) on a single node with eight H100 GPUs. For S^{3}R, both models use a draft length of $k = 6$, a sparse budget ratio of $s = 0.05$, and the KV-aware AIMD scheduler in Appendix B.2. Full training and system configurations are provided in Appendix C.

Baselines. We compare S^{3}R against dense **AR** decoding and three representative speculative proposal strategies. **N-gram Lookup** uses prompt lookup decoding (Saxena, 2023) to propose continuations from matching token suffixes within the current prompt or response. **History Retrieval** follows RhymeRL (He et al., 2025) and retrieves draft continuations from previous responses to the same prompt. **Auxiliary Proposer** uses a pretrained EAGLE3 network (Li et al., 2025b) with $k = 5$, keeping its weights frozen as the target policy evolves. The main comparison uses no additional drafter training; online co-training is examined in Appendix D. Baseline implementation details are given in Appendix C.4.

Metrics. We report rollout generation time, end-to-end training-step time, and training throughput (tokens/s per GPU), with speedups relative to AR. Rollout time covers response generation, while step time also includes reward computation, actor updates, and system overhead. Block efficiency τ is the average number of output tokens per speculative round, including the bonus token. We evaluate model quality using AIME26 and ArXiv-Math pass@1 scores from the best checkpoint. Measurement and reporting details are provided in Appendix C.3.

6.2 End-to-End Acceleration with Comparable Task Performance

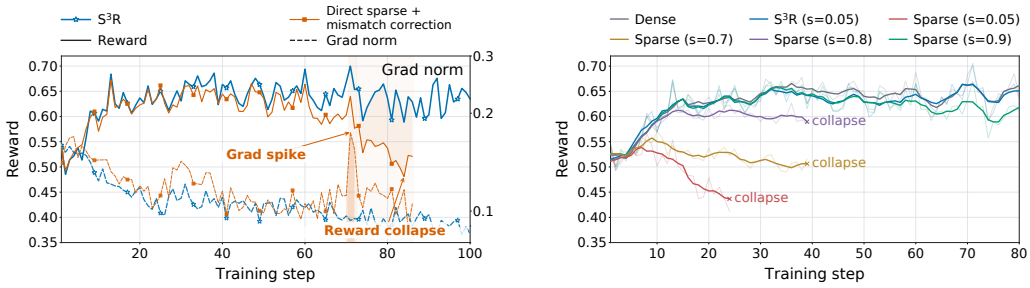
Table 1: Main results on Qwen3-1.7B and Qwen3-8B. Benchmark scores are pass@1 (%) from the best checkpoint. Timing and throughput cover the full training run after the first warm-up step. Values use one decimal place; bold and underlined entries mark the best and second-best results before rounding within each model size.

Model	Method	Benchmarks		Efficiency			
		AIME26	ArXiv-Math	τ	Rollout (min) ↓	Step (min) ↓	Throughput/Speedup
Qwen3-1.7B	AR	**37.3**	**10.0**	N/A	10.5 (base)	13.9 (base)	2308.6 / 1.0×
	N-gram Lookup	<u>37.1</u>	9.8	1.3	12.5 (+19.1%)	19.0 (+36.3%)	1671.0 / 0.7×
	History Retrieval	36.0	9.8	<u>2.0</u>	10.4 (−1.7%)	13.3 (−4.2%)	2412.7 / 1.0×
	Auxiliary Proposer	36.7	<u>9.9</u>	1.9	<u>9.1 (−13.9%)</u>	<u>12.4 (−10.5%)</u>	<u>2562.0 / 1.1×</u>
	Ours	<u>37.1</u>	<u>9.9</u>	**5.6**	**6.5 (-38.6%)**	**9.9 (-28.5%)**	**3213.8 / 1.4×**
Qwen3-8B	AR	65.4	15.7	N/A	36.1 (base)	50.2 (base)	964.8 / 1.0×
	N-gram Lookup	64.0	16.1	1.3	38.8 (+7.5%)	55.0 (+9.6%)	876.5 / 0.9×
	History Retrieval	<u>66.0</u>	<u>16.4</u>	<u>2.0</u>	36.1 (0.0%)	49.8 (−0.8%)	959.3 / 1.0×
	Auxiliary Proposer	**66.3**	**16.8**	1.9	<u>31.6 (−12.5%)</u>	<u>45.6 (−9.2%)</u>	<u>1055.5 / 1.1×</u>
	Ours	65.4	16.3	**5.5**	**24.1 (-33.2%)**	**37.4 (-25.5%)**	**1302.6 / 1.4×**

Table 1 shows that S^{3}R achieves the highest end-to-end training throughput among the compared methods while maintaining comparable task performance. Specifically, it reaches $1.39\times$ and $1.35\times$ the throughput of AR on Qwen3-1.7B and Qwen3-8B, respectively. In contrast, N-gram Lookup reduces training throughput because its low block efficiency ($\tau \approx 1.3$) limits the amortization of the verification cost. Although History Retrieval improves block efficiency, it provides little overall acceleration. We hypothesize that, at a decoding temperature of 1.0, limited reuse of reasoning continuations constrains both lookup-based methods, while history maintenance introduces additional overhead. The frozen Auxiliary Proposer provides modest throughput gains, but its declining acceptance rate (Figure 5c) suggests a growing mismatch between the frozen proposer and the evolving policy. Nevertheless, the Auxiliary Proposer achieves higher throughput than History Retrieval despite having lower block efficiency, indicating that overall throughput depends on both block efficiency and proposal-generation and runtime costs. S^{3}R maintains high block efficiency ($\tau \approx 5.5$) at $s = 0.05$ by using proposals from the current policy, without requiring historical coverage or separate drafter adaptation. This high block efficiency amortizes the cost of dense verification, while sparse drafting reduces KV reads in our attention-dominated workloads. Finally, the reward and response-length trajectories of S^{3}R closely match those of AR (Appendix E.1), which is consistent with preserving the dense response-group distribution used for learning.

6.3 Stabilizing Training under Aggressive Sparsity

Gradient instability despite training-side correction. Figure 4a compares S^{3}R with *Direct sparse + mismatch correction*, which removes speculative verification and applies sequence-level filtering and tokenwise importance reweighting to sparse rollouts (following Luo et al. (2026)). Both use $s = 0.05$ with matched model, data, optimizer, and batching settings. Although their early rewards are similar, the corrected sparse variant already exhibits larger gradient-norm fluctuations, followed by a sharp spike around step 70 and further reward deterioration. S^{3}R avoids this spike-and-collapse pattern throughout the 100-step run. Similar mean rewards can therefore mask differences in update stability. This failure is consistent with the concern in Section 3.2: correcting a response's likelihood does not generally restore the reward comparisons within its GRPO group. The comparison supports addressing this mismatch at sampling time, before rewards define the group-relative advantages.



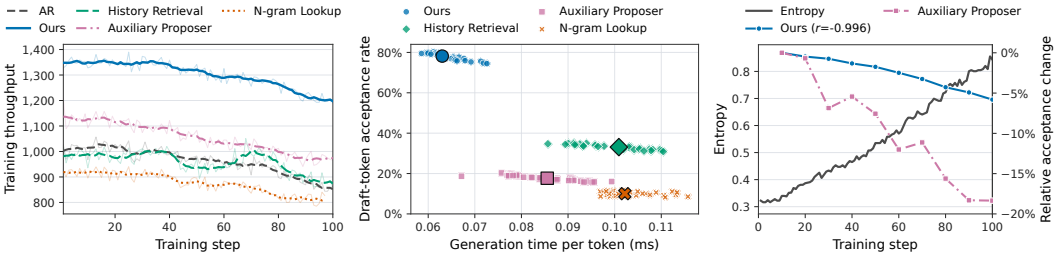
(a) Verification versus mismatch correction.

(b) Stability across sparse budgets.

Figure 4: Training stability under sparse rollout variants. (a) Reward and gradient norm for S^{3}R and Direct sparse + mismatch correction; (b) reward across sparse budgets.

Verification enables stable training at aggressive sparsity. Figure 4b compares Dense and S^{3}R with direct sparse rollouts that use neither speculative verification nor training-side mismatch correction. Direct sparse rollouts collapse early at $s = 0.05$ and remain unstable at $s = 0.7$ and $s = 0.8$. Only $s = 0.9$ sustains relatively stable learning over the displayed window, indicating that stability constrains how aggressively these direct rollouts can reduce KV reads. This sensitivity agrees qualitatively with Zhou et al. (2026b), who link stable sparse-rollout RL to sufficient lower-tail alignment with the dense policy. S^{3}R follows the Dense reward trajectory at $s = 0.05$. Here, sparsity is confined to proposals, with full-attention verification and residual sampling preserving the dense rollout distribution. Together with Proposition 2, these results support selecting the drafting budget for efficiency without introducing sparsity-induced changes to the learning signal.

6.4 Analysis of RL Rollout Efficiency



(a) End-to-end training throughput.

(b) Timing–acceptance trade-off.

(c) Entropy and acceptance drift.

Figure 5: Qwen3-8B rollout diagnostics during RL training. (a) End-to-end training throughput. (b) Generation time per processed token (prompt and response) versus draft-token acceptance; outlined markers denote method medians. (c) Policy entropy and relative change in draft-token acceptance.

Sustained efficiency during RL training. Figure 5a shows that S^{3}R maintains higher end-to-end training throughput than the evaluated baselines as training progresses, including as Qwen3-8B responses grow longer (Appendix E.1). In Figure 5b, S^{3}R combines 78.2% median draft-token acceptance with the lowest median generation time, 0.0630 ms per processed token, among the evaluated speculative methods. History Retrieval attains higher acceptance than Auxiliary Proposer but incurs greater generation cost, showing why acceptance alone cannot rank acceleration methods. In long-context RLVR, each accepted block must repay the cost of drafting, verification, and state management (Section 5). S^{3}R benefits from reducing historical KV reads during drafting in the attention-dominated regime identified in Figure 1a, while retaining enough accepted progress to amortize dense verification. The Qwen3-1.7B results exhibit the same pattern in Appendix E.3.

Proposal alignment as the policy evolves. Figure 5c shows a smaller relative decline in block efficiency for S^{3}R than for the frozen Auxiliary Proposer, alongside rising policy entropy. On Qwen3-8B, S^{3}R's overall acceptance decreases relatively 5%; later draft positions exhibit larger declines (Appendix E.2). Higher entropy reflects greater uncertainty in token selection during RL

exploration, while ongoing policy updates can leave a frozen drafter stale. By sharing the current rollout-policy weights, S^{3}R avoids this parameter lag, although sparse approximation can still affect acceptance. Together with the checkpoint-portability results in Appendix D, these findings support S^{3}R's ability to generate useful proposals across policy updates without separate drafter training.

Balancing draft cost and accepted progress. We sweep the sparse drafting budget with full-attention verification enabled, measuring median generation speedup over AR in the same engine (Figure 6). Increasing s from 0.05 to 0.50 raises acceptance from 80.4% to 96.7%, while speedup falls from 1.370× to 1.058×. Retaining more context improves proposal alignment, but the extra KV reads cost more than the additional accepted tokens save, as anticipated by Eq. 5. Reducing s to 0.03 lowers both acceptance and speedup, supporting our default $s = 0.05$ as a practical balance in this workload. Exact speculative sampling makes this an efficiency trade-off: the dense response-group distribution is preserved across budgets. Appendix F.2.3 reports sweeps for both model sizes, and Appendix F.3 provides a calibrated cost interpretation.

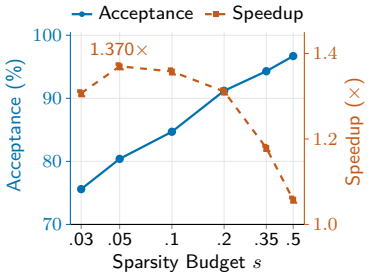


Figure 6: Qwen3-8B budget sweep at a frozen step-60 checkpoint ($B = 64$, initial context 8K, $k = 6$).

6.5 Managing Full-KV State under Memory Pressure

Table 2: KV-aware AIMD scheduler ablation on Qwen3-1.7B, aggregated over 10 RL steps after warm-up. (a) Generation throughput across all methods. (b) KV overhead with fixed concurrency and the KV-aware AIMD scheduler, which adaptively adjusts the concurrency limit between 160 and 256. All non-AR variants use sparse drafting and full-attention verification.

(a) Generation throughput

Method	Tok/s/GPU ↑	Gain
AR	3,989.0	Ref.
Recompute-256	5,894.9	+47.8%
Offload-128	5,411.8	+35.7%
Offload-160	5,699.8	+42.9%
Offload-256	6,014.9	+50.8%
AIMD scheduler	**6,362.1**	**+59.5%**

(b) KV overhead

Metric ↓	Offload-256	AIMD scheduler
Peak CPU KV-cache occupancy (pages)	52,464	21,358
Offload episodes	1,097	59
Reoffload (%)	5.0	0.0
Move/1K (pages)	18.0	1.3

Sparse self-drafting reduces KV reads while retaining the complete historical state required for faithful verification. As long responses accumulate, this state can exceed GPU capacity, making its management part of realizing the decoding gain. In Table 2, replacing recomputation with offloading at a concurrency ceiling of 256 improves generation throughput by 2.0%. AIMD adds a further 5.8% over Offload-256, bringing the gain over AR to 59.5%, while reducing transferred pages per 1,000 generated tokens from 18.0 to 1.3. The throughput improvement alongside reduced KV transfers is consistent with less KV-cache thrashing as the working set grows. Offload-128 and Offload-160 both underperform Offload-256: a permanently conservative limit also sacrifices useful batching when memory is available. Adapting admission to KV pressure and restoration demand therefore helps retain rollout parallelism while preserving the full state needed for dense verification.

7 Conclusion and Future Work

We introduced S^{3}R, which accelerates long-context RLVR through sparse self-drafting and exact speculative sampling. Under a shared rollout snapshot, it preserves the dense response-group distribution without requiring separate drafter training. With KV-aware runtime support, S^{3}R achieves 1.39× and 1.35× the end-to-end training throughput of AR on Qwen3-1.7B and Qwen3-8B, respectively, while maintaining stable training and comparable task performance. Future work will explore adaptive sparse budgets and context selection, along with extensions to MoE and linear-attention models. Another direction is to coordinate KV management with group completion and rollout–optimization transitions to reduce overhead while preserving exact sampling.

AI USE STATEMENT

We used generative AI tools to find related work on sparse RL rollouts and speculative decoding, and to draft and polish descriptions of S^3R's sparse self-drafting, dense verification, and runtime support. AI also assisted with deriving and checking mathematical claims about the limitations of response-wise importance correction in GRPO and the preservation of dense response-group distributions under exact speculative sampling, as well as the latency trade-off between drafting cost and accepted progress. The authors reviewed all AI-assisted material and take full responsibility for the paper.

REFERENCES

Sergios Agapiou, Omiros Papaspiliopoulos, Daniel Sanz-Alonso, and Andrew M. Stuart. Importance sampling: Intrinsic dimension and computational cost. *Statistical Science*, 32(3):405–431, 2017. doi: 10.1214/17-STS611. URL https://arxiv.org/abs/1511.06196.

Zefan Cai, Wen Xiao, Hanshi Sun, Cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, Zhen Dong, Anima Anandkumar, Abedelkadir Asi, and Junjie Hu. R-KV: redundancy-aware KV cache compression for training-free reasoning models acceleration. *CoRR*, abs/2505.24133, 2025. doi: 10.48550/ARXIV.2505.24133. URL https://doi.org/10.48550/arXiv.2505.24133.

Chi-Chih Chang, Siqi Zhu, Zhichen Zeng, Haibin Lin, Jiaxuan You, Mohamed S. Abdelfattah, Ziheng Jiang, and Xuehai Qian. SRT: accelerating reinforcement learning via speculative rollout with tree-structured cache. *CoRR*, abs/2601.09083, 2026. doi: 10.48550/ARXIV.2601.09083. URL https://doi.org/10.48550/arXiv.2601.09083.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *CoRR*, abs/2302.01318, 2023. doi: 10.48550/ARXIV.2302.01318. URL https://doi.org/10.48550/arXiv.2302.01318.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding, 2026a. URL https://arxiv.org/abs/2602.06036.

Longze Chen, Renke Shan, Huiming Wang, Lu Wang, Ziqiang Liu, Run Luo, Jiawei Wang, Hamid Alinejad-Rokny, and Min Yang. CLaSp: In-context layer skip for self-speculative decoding. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 31608–31618. Association for Computational Linguistics, 2025a. doi: 10.18653/V1/2025.ACL-LONG.1525. URL https://doi.org/10.18653/v1/2025.acl-long.1525.

Qiaoling Chen, Zijun Liu, Peng Sun, Shenggui Li, Guoteng Wang, Ziming Liu, Yonggang Wen, Siyuan Feng, and Tianwei Zhang. ReSpec: Towards optimizing speculative decoding in reinforcement learning systems. *CoRR*, abs/2510.26475, 2025b. doi: 10.48550/ARXIV.2510.26475. URL https://doi.org/10.48550/arXiv.2510.26475.

Qiaoling Chen, Zhisheng Ye, Tian Tang, Peng Sun, Boyu Tian, Guoteng Wang, Shenggui Li, Yonggang Wen, Zhenhua Han, and Tianwei Zhang. CONCUR: High-throughput agentic batch inference of LLM via congestion-based concurrency control. *CoRR*, abs/2601.22705, 2026b. doi: 10.48550/arXiv.2601.22705. URL https://arxiv.org/abs/2601.22705.

Zhuoming Chen, Hongyi Liu, Yang Zhou, Haizhong Zheng, and Beidi Chen. Jackpot: Optimal budgeted rejection sampling for extreme actor-policy mismatch reinforcement learning. *CoRR*, abs/2602.06107, 2026c. doi: 10.48550/ARXIV.2602.06107. URL https://doi.org/10.48550/arXiv.2602.06107.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, Qinyu Chen, Chengqi Deng, Jiping Yu, Damai Dai, Zhengyan Zhang, Yixuan Wei, Yixuan Tan, Wenkai Yang, Runxin Xu, Yu Wu, Zhean Xu, Xuanyu Wang, Muyang

Chen, Rui Tian, Xiao Bi, Zhewen Hao, Shaoyuan Chen, Huanqi Cao, Wentao Zhang, Anyi Xu, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. DSpark: Confidence-scheduled speculative decoding with semi-autoregressive generation, 2026. URL https://arxiv.org/abs/2607.05147.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *CoRR*, abs/2501.12948, 2025. doi: 10.48550/ARXIV.2501.12948. URL https://doi.org/10.48550/arXiv.2501.12948.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin T. Vechev. Beyond benchmarks: MathArena as an evaluation platform for mathematics with LLMs. *CoRR*, abs/2605.00674, 2026. doi: 10.48550/ARXIV.2605.00674. URL https://doi.org/10.48550/arXiv.2605.00674.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed A. Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling early exit inference and self-speculative decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 12622–12642. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.681. URL https://doi.org/10.18653/v1/2024.acl-long.681.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, Tongkai Yang, Binhang Yuan, and Yi Wu. AREAL: A large-scale asynchronous reinforcement learning system for language reasoning. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, pp. 36256–36282. Curran Associates, Inc., 2025. doi: 10.52202/085713-1218. URL https://papers.nips.cc/paper_files/paper/2025/hash/33c00862bfa29ac72ecf630a41e19352-Abstract-Conference.html.

Hao Gu, Hao Wang, Jiacheng Liu, Lujun Li, Qiyuan Zhu, Bei Liu, Binxing Xu, Lei Wang, Xintong Yang, Sida Lin, Sirui Han, and Yike Guo. QaRL: Rollout-aligned quantization-aware RL for fast and stable training under training-inference mismatch. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics, ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 17051–17064. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.FINDINGS-ACL.841. URL https://doi.org/10.18653/v1/2026.findings-acl.841.

Jingkai He, Tianjian Li, Erhu Feng, Dong Du, Qian Liu, Tao Liu, Yubin Xia, and Haibo Chen. History rhymes: Accelerating LLM reinforcement learning with RhymeRL. *CoRR*, abs/2508.18588, 2025. doi: 10.48550/ARXIV.2508.18588. URL https://doi.org/10.48550/arXiv.2508.18588.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D. Lee, and Di He. REST: retrieval-based speculative decoding. In Kevin Duh, Helena Gómez-Adorno, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pp. 1582–1595. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.NAACL-LONG.88. URL https://doi.org/10.18653/v1/2024.naacl-long.88.

Qinghao Hu, Shang Yang, Junxian Guo, Xiaozhe Yao, Yujun Lin, Yuxian Gu, Han Cai, Chuang Gan, Ana Klimovic, and Song Han. Taming the long-tail: Efficient reasoning RL training with adaptive drafter. In Benjamin C. Lee, Harry Xu, Mark Silberstein, and Bingyao Li (eds.), *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2026, Pittsburgh, PA, USA, March 22-26, 2026*, pp. 1933–1948. ACM, 2026. doi: 10.1145/3779212.3790231. URL https://doi.org/10.1145/3779212.3790231.

Hayate Iso, Tiyasa Mitra, Sudipta Mondal, Rasoul Shafipour, Venmugil Elango, Terry Kong, Yuki Huang, Seonjin Na, Izzy Putterman, Benjamin Chislett, Maor Ashkenazi, Joseph Guman, Gerald Shen, Tugrul Konuk, Ashwath Aithal, Ritika Borkar, Ran Zilberstein, and Bita Rouhani. Accelerating RL post-training rollouts via system-integrated speculative decoding. *CoRR*, abs/2604.26779, 2026. doi: 10.48550/ARXIV.2604.26779. URL https://doi.org/10.48550/arXiv.2604.26779.

Minseo Kim, Minjae Lee, Seunghyuk Oh, Kevin Galim, Donghoon Kim, Coleman Hooper, Harman Singh, Amir Gholami, Hyung Il Koo, and Wonjun Kang. EfficientRollout: System-aware self-speculative decoding for RL rollouts. *CoRR*, abs/2606.18967, 2026. doi: 10.48550/ARXIV.2606.18967. URL https://doi.org/10.48550/arXiv.2606.18967.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In Jason Flinn, Margo I. Seltzer, Peter Druschel, Antoine Kaufmann, and Jonathan Mace (eds.), *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pp. 611–626. ACM, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Yingru Li, Jiacai Liu, Jiawei Xu, Yuxuan Tong, Ziniu Li, and Baoxiang Wang. Trust region masking for long-horizon LLM reinforcement learning. *CoRR*, abs/2512.23075, 2025a. doi: 10.48550/ARXIV.2512.23075. URL https://doi.org/10.48550/arXiv.2512.23075.

Yuhang Li, Reena Elangovan, Xin Dong, Priyadarshini Panda, and Brucek Khailany. QuRL: Efficient reinforcement learning with quantized rollout. *CoRR*, abs/2602.13953, 2026. doi: 10.48550/ARXIV.2602.13953. URL https://doi.org/10.48550/arXiv.2602.13953.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In Amir Globerson, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, pp. 22947–22970. Curran Associates, Inc., 2024. doi: 10.52202/079017-0722. URL https://papers.nips.cc/paper_files/paper/2024/hash/28ab418242603e0f7323e54185d19bde-Abstract-Conference.html.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: scaling up inference acceleration of large language models via training-time test. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, pp. 136737–136756. Curran Associates, Inc., 2025b. doi: 10.52202/085713-4562. URL https://papers.nips.cc/paper_files/paper/2025/hash/c7b5a35ea98b62512a869c19ea7b03cb-Abstract-Conference.html.

Ling Team and Inclusion AI. Every step evolves: Scaling reinforcement learning for trillion-scale thinking model. *CoRR*, abs/2510.18855, 2025. doi: 10.48550/ARXIV.2510.18855. URL https://doi.org/10.48550/arXiv.2510.18855.

Bingshuai Liu, Ante Wang, Zijun Min, Liang Yao, Haibo Zhang, Yang Liu, Anxiang Zeng, and Jinsong Su. SPEC-RL: accelerating on-policy reinforcement learning via speculative rollouts. *CoRR*, abs/2509.23232, 2025. doi: 10.48550/ARXIV.2509.23232. URL https://doi.org/10.48550/arXiv.2509.23232.

Xiaoxuan Liu, Jiaxiang Yu, Jongseok Park, Ion Stoica, and Alvin Cheung. Speculative decoding: Performance or illusion? *CoRR*, abs/2601.11580, 2026a. doi: 10.48550/ARXIV.2601.11580. URL https://doi.org/10.48550/arXiv.2601.11580.

Yuesong Liu, Yuan Zeng, Min Lyu, Ruilin Liu, Yu Guo, and Yinlong Xu. A sparse glimpse of the whole: Train-free self-speculative decoding. *CoRR*, abs/2607.27735, 2026b. doi: 10.48550/ARXIV.2607.27735. URL https://doi.org/10.48550/arXiv.2607.27735.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. DeepScaleR: Surpassing O1-Preview with a 1.5B model by scaling RL. Notion Blog, 2025. URL https://pretty-radio-b75.notion.site/19681902c1468005bed8ca303013a4e2.

Sijia Luo, Xiaokang Zhang, Yuxuan Hu, Bohan Zhang, Ke Wang, Jinbo Su, Mengshu Sun, Lei Liang, and Jing Zhang. Sparse-RL: Breaking the memory wall in LLM reinforcement learning via stable sparse rollouts. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 43188–43201. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.ACL-LONG.2000. URL https://doi.org/10.18653/v1/2026.acl-long.2000.

OpenAI. OpenAI o1 system card. *CoRR*, abs/2412.16720, 2024. doi: 10.48550/ARXIV.2412.16720. URL https://doi.org/10.48550/arXiv.2412.16720.

Penghui Qi, Zichen Liu, Xiangxin Zhou, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Defeating the training-inference mismatch via FP16. *CoRR*, abs/2510.26788, 2025. doi: 10.48550/ARXIV.2510.26788. URL https://doi.org/10.48550/arXiv.2510.26788.

Ruoyu Qin, Weiran He, Weixiao Huang, Yangkun Zhang, Yikai Zhao, Bo Pang, Xinran Xu, Yingdi Shan, Yongwei Wu, and Mingxing Zhang. Seer: Online context learning for fast synchronous LLM reinforcement learning. *CoRR*, abs/2511.14617, 2025. doi: 10.48550/ARXIV.2511.14617. URL https://doi.org/10.48550/arXiv.2511.14617.

Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian En-Hsu Yen, Avner May, Tianqi Chen, and Beidi Chen. MagicDec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=CS2JWaziYr.

Apoorv Saxena. Prompt lookup decoding, November 2023. URL https://github.com/apoorvumang/prompt-lookup-decoding/.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. doi: 10.48550/arXiv.1707.06347. URL https://arxiv.org/abs/1707.06347.

Zelei Shao, Vikranth Srivatsa, Sanjana Srivastava, Qingyang Wu, Alpay Ariyak, Xiaoxia Wu, Ameen Patel, Jue Wang, Percy Liang, Tri Dao, Ce Zhang, Yiying Zhang, Ben Athiwaratkun, Chenfeng Xu, and Junxiong Wang. Beat the long tail: Distribution-aware speculative decoding for RL training. *CoRR*, abs/2511.13841, 2025. doi: 10.48550/ARXIV.2511.13841. URL https://doi.org/10.48550/arXiv.2511.13841.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300, 2024. doi: 10.48550/ARXIV.2402.03300. URL https://doi.org/10.48550/arXiv.2402.03300.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*, pp. 1279–1297. ACM, 2025. doi: 10.1145/3689031.3696075. URL https://doi.org/10.1145/3689031.3696075.

Mingbo Song, Heming Xia, Jun Zhang, Chak Tou Leong, Qiancheng Xu, Wenjie Li, and Sujian Li. KNN-SSD: enabling dynamic self-speculative decoding via nearest neighbor layer set optimization. In Vera Demberg, Kentaro Inui, and Lluís Marquez (eds.), *Findings of the Association for Computational Linguistics: EACL 2026, Rabat, Morocco, March 24-29, 2026*, Findings of ACL, pp. 641–655. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.FINDINGS-EACL.31. URL https://doi.org/10.18653/v1/2026.findings-eacl.31.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. ShadowKV: KV cache in shadows for high-throughput long-context LLM inference. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*, pp. 57355–57373. PMLR, 2025. URL https://proceedings.mlr.press/v267/sun25b.html.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: query-aware sparsity for efficient long-context LLM inference. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 47901–47911. PMLR, 2024. URL https://proceedings.mlr.press/v235/tang24l.html.

Rishabh Tiwari, Haocheng Xi, Aditya Tomar, Coleman Richard Charles Hooper, Sehoon Kim, Maxwell Horton, Mahyar Najibi, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. QuantSpec: Self-speculative decoding with hierarchical quantized KV cache. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*, pp. 59668–59686. PMLR, 2025. URL https://proceedings.mlr.press/v267/tiwari25b.html.

Boxin Wang, Chankyu Lee, Nayeon Lee, Sheng-Chieh Lin, Wenliang Dai, Yang Chen, Yangyi Chen, Zhuolin Yang, Zihan Liu, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Nemotron-Cascade: Scaling cascaded reinforcement learning for general-purpose reasoning models, 2025a. URL https://arxiv.org/abs/2512.13607.

Junxiong Wang, Fengxiang Bie, Jisen Li, Zhongzhu Zhou, Zelei Shao, Yubo Wang, Yinghui Liu, Qingyang Wu, Avner May, Sri Yanamandra, Yineng Zhang, Ce Zhang, Tri Dao, Percy Liang, Ben Athiwaratkun, Shuaiwen Leon Song, Chenfeng Xu, and Xiaoxia Wu. When RL meets adaptive speculative training: A unified training-serving system. *CoRR*, abs/2602.06932, 2026a. doi: 10.48550/ARXIV.2602.06932. URL https://doi.org/10.48550/arXiv.2602.06932.

Siqi Wang, Hailong Yang, Junjie Zhu, Xuezhu Wang, Yufan Xu, and Depei Qian. RLHF-Spec: Breaking the efficiency bottleneck in RLHF training via adaptive drafting. *CoRR*, abs/2512.04752, 2025b. doi: 10.48550/ARXIV.2512.04752. URL https://doi.org/10.48550/arXiv.2512.04752.

Zili Wang, Zhaopeng Qiu, Yuekai Zhang, Shuang Yu, and Junjie Lai. Online draft co-training for speculative decoding in large-scale, long-context RL post-training, 2026b. URL https://arxiv.org/abs/2609.07108.

Samuel Williams, Andrew Waterman, and David A. Patterson. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, 2009. doi: 10.1145/1498765.1498785. URL https://doi.org/10.1145/1498765.1498785.

Heming Xia, Yongqi Li, Jun Zhang, Cunxiao Du, and Wenjie Li. SWIFT: on-the-fly self-speculative decoding for LLM inference acceleration. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=EKJhH5D5wA.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: an open-source LLM reinforcement learning system at scale. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, pp. 113222–113244. Curran Associates, Inc., 2025. doi: 10.52202/085713-3775. URL https://papers.nips.cc/paper_files/paper/2025/hash/a4277440d50f1f15d2cb4c14f7e0c0d2-Abstract-Conference.html.

Yikang Yue, Yuqi Xue, and Jian Huang. Vegas: Self-speculative decoding with verification-guided sparse attention. *CoRR*, abs/2602.07223, 2026. doi: 10.48550/ARXIV.2602.07223. URL https://doi.org/10.48550/arXiv.2602.07223.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & Verify: Lossless large language model acceleration via self-speculative decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 11263–11282. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.607. URL https://doi.org/10.18653/v1/2024.acl-long.607.

Yiqi Zhang, Huiqiang Jiang, Xufang Luo, Zhihe Yang, Chengruidong Zhang, Yifei Shen, Dongsheng Li, Yuqing Yang, Lili Qiu, and Yang You. SortedRL: Accelerating RL training for LLMs through online length-aware scheduling. *CoRR*, abs/2603.23414, 2026a. doi: 10.48550/ARXIV.2603.23414. URL https://doi.org/10.48550/arXiv.2603.23414.

Yizhou Zhang, Ning Lv, Teng Wang, and Jisheng Dang. FastGRPO: Accelerating policy optimization via concurrency-aware speculative decoding and online draft learning. *CoRR*, abs/2509.21792, 2025. doi: 10.48550/ARXIV.2509.21792. URL https://doi.org/10.48550/arXiv.2509.21792.

Yuheng Zhang, Chenlu Ye, Shuowei Jin, Changlong Yu, Wei Xiong, Saurabh Sahu, and Nan Jiang. Rethinking importance sampling in LLM policy optimization: A cumulative token perspective. *CoRR*, abs/2605.07331, 2026b. doi: 10.48550/ARXIV.2605.07331. URL https://doi.org/10.48550/arXiv.2605.07331.

Yilong Zhao, Jiaming Tang, Kan Zhu, Zihao Ye, Chi-Chih Chang, Chaofan Lin, Jongseok Park, Guangxuan Xiao, Mohamed S. Abdelfattah, Mingyu Gao, Baris Kasikci, Song Han, and Ion Stoica. Accelerating large-scale reasoning model inference with sparse self-speculative decoding. *CoRR*, abs/2512.01278, 2025. doi: 10.48550/ARXIV.2512.01278. URL https://doi.org/10.48550/arXiv.2512.01278.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. *CoRR*, abs/2507.18071, 2025. doi: 10.48550/ARXIV.2507.18071. URL https://doi.org/10.48550/arXiv.2507.18071.

Jiajun Zhou, Wei Shao, Lingchao Zheng, Yuwei Fan, and Ngai Wong. AIS: adaptive importance sampling for quantized RL. *CoRR*, abs/2605.13907, 2026a. doi: 10.48550/ARXIV.2605.13907. URL https://doi.org/10.48550/arXiv.2605.13907.

Yang Zhou, Ranajoy Sadhukhan, Zhaofeng Sun, Zhuoming Chen, Souvik Kundu, Saket Dingliwal, Sai Muralidhar Jayanthi, Aram Galstyan, Haizhong Zheng, and Beidi Chen. Sparrow: Sparse rollout for stable and efficient long-context RL of large language models. *CoRR*, abs/2606.08446, 2026b. doi: 10.48550/ARXIV.2606.08446. URL https://doi.org/10.48550/arXiv.2606.08446.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. DistillSpec: Improving speculative decoding via knowledge distillation. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=rsY6J3ZaTF.

Yuzhen Zhou, Jiajun Li, Yusheng Su, Gowtham Ramesh, Zilin Zhu, Xiang Long, Chenyang Zhao, Jin Pan, Xiaodong Yu, Ze Wang, Kangrui Du, Jialian Wu, Ximeng Sun, Jiang Liu, Qiaolin Yu, Hao Chen, Zicheng Liu, and Emad Barsoum. APRIL: active partial rollouts in reinforcement learning to tame long-tail generation. *CoRR*, abs/2509.18521, 2025. doi: 10.48550/ARXIV.2509.18521. URL https://doi.org/10.48550/arXiv.2509.18521.

A Policy Mismatch and Distribution Preservation

This appendix analyzes training-side correction and distribution preservation under sparse proposals. Section A.1 relates the token-level ratio in Eq. (1) to complete-response importance weights. Section A.2 proves Proposition 1 through a change-of-measure identity and an explicit GRPO counterexample. Section A.3 proves Proposition 2 from conditional token sampling through complete responses and independent response groups.

We use the notation in Section 3.1 and fix the rollout snapshot θ_{old}. For a fixed response index i, we write $w_{s,t}$ for the sparsity ratio w_s evaluated at $(o_t^i \mid x, o_{<t}^i)$. All policy probabilities include the corresponding sampling transforms.

A.1 Tokenwise and response-wise importance weights

For a fixed direct-sparse rollout procedure, let $h_t^i = (x, o_{<t}^i)$ and interpret $\pi_{\theta_{\text{old}},s}^{\text{sparse}}(\cdot \mid h_t^i)$ as its prefix-conditional token law. Equation (1) gives the ratio at one sampled token:

$$\hat{w} = \underbrace{\frac{\pi_\theta(o_t^i \mid h_t^i)}{\pi_{\theta_{\text{old}}}^{\text{dense}}(o_t^i \mid h_t^i)}}_{w(\theta)} \underbrace{\frac{\pi_{\theta_{\text{old}}}^{\text{dense}}(o_t^i \mid h_t^i)}{\pi_{\theta_{\text{old}},s}^{\text{sparse}}(o_t^i \mid h_t^i)}}_{w_{s,t}}. \tag{6}$$

The first factor accounts for the learner's change from the dense rollout snapshot; the second corrects the current token conditional at h_t^i. A single $w_{s,t}$ leaves the sparse distribution of the prefix and continuation uncorrected. CTPO addresses prefix state–action mismatch through cumulative token importance ratios (Zhang et al., 2026b). Our analysis additionally examines the dependence of a response's GRPO advantage on other sampled group members.

Sparse-RL combines response filtering with tokenwise reweighting (Luo et al., 2026), while Sparrow pairs mismatch-guided sparsity scheduling with truncated importance sampling (Zhou et al., 2026b). For other rollout mismatches, IcePop masks tokens with large inference–training probability discrepancies (Ling Team & Inclusion AI, 2025), and AIS adapts importance correction for quantized rollouts (Zhou et al., 2026a). These interventions mitigate mismatch without generally restoring the dense response-group law. To isolate the role of group-relative advantages in direct-sparse rollout, we grant each complete response its exact importance weight.

Let P and Q_s be the dense and direct-sparse laws defined in Section 3.2. If $Q_s(o \mid x) > 0$ whenever $P(o \mid x) > 0$, the chain rule gives the exact response weight

$$W_i = \frac{P(o^i \mid x)}{Q_s(o^i \mid x)} = \prod_{t=1}^{|o^i|} w_{s,t}. \tag{7}$$

The standard importance-sampling identity (Agapiou et al., 2017) gives $\mathbb{E}_{Q_s}[W_i f(o^i)] = \mathbb{E}_P[f(o^i)]$ for any integrable function f of one complete response. For groups of independently sampled responses, the joint likelihood ratio is $\prod_{j=1}^G W_j$. Weighting only response i's update by W_i leaves its group members under Q_s, which can bias the GRPO update even with exact response weights (Section A.2). S^{3}R instead preserves the dense response-group law before computing rewards and advantages so that its policy importance ratio remains $w(\theta)$ (Section A.3).

A.2 Limits of response-wise correction in GRPO

Proof of Proposition 1. Fix the prompt x and learner parameters. Suppressing the conditioning on x, let $P(u)$ be the dense rollout probability of a complete response u, and $Q_s(u)$ the probability obtained by sampling the sparse drafter directly. The corresponding exact sparse-to-dense weight is $W(u) = P(u)/Q_s(u)$, with $P(u) > 0$ implying $Q_s(u) > 0$ and all expectations finite. Here an update is the group-averaged surrogate gradient. Both groups use the same GRPO surrogate referenced to the dense rollout policy; W multiplies each response gradient as a fixed outer factor.

For two independent responses U, V, let $g(u, v)$ be the gradient contribution of u after its advantage is computed from the pair (u, v). The group gradient averages $g(u, v)$ and $g(v, u)$. By exchangeability, its expected value equals the expected contribution of the first response, also when each

contribution is weighted by its own W. Let $H(v) = \sum_u P(u)g(u, v)$ be the expected gradient of a focal response drawn from the dense law P when its comparison is fixed at v. Write G_{corr} for the expected gradient after weighting direct-sparse groups response by response, and G_{dense} for the dense-rollout baseline. Then

$$\begin{aligned} G_{\text{corr}} &= \sum_{u,v} Q_s(u)Q_s(v)W(u)g(u, v) \\ &= \sum_{u,v} P(u)Q_s(v)g(u, v) = \sum_v Q_s(v)H(v), \\ G_{\text{dense}} &= \sum_{u,v} P(u)P(v)g(u, v) = \sum_v P(v)H(v). \end{aligned} \tag{8}$$

In the weighted expectation, $W(u)$ replaces the focal factor $Q_s(u)$ with $P(u)$. The focal response is thus accounted for under the dense rollout law P, while its comparison response still follows the direct-sparse law Q_s. For larger groups, the other $G - 1$ responses likewise retain this sparse law. A difference in expected gradients occurs when $H(v)$ varies with the comparison response and the shift from the dense law P to the direct-sparse law Q_s does not cancel.

A GRPO counterexample. Consider three one-token responses a, b, c with rewards $(0, 1, 2)$. In (a, b, c) order, take the dense law $P = (1/3, 1/3, 1/3)$ and the direct-sparse law $Q_s = (1/2, 1/3, 1/6)$, giving exact response weights $(W(a), W(b), W(c)) = (2/3, 1, 2)$. For two-response groups, the sample standard deviation of the rewards is $\sigma(u, v) = |r(u) - r(v)|/\sqrt{2}$. With stabilizer $\varepsilon > 0$, the group-normalized advantage of u is

$$A(u, v) = \frac{r(u) - [r(u) + r(v)]/2}{\sigma(u, v) + \varepsilon} = \frac{r(u) - r(v)}{\sqrt{2}|r(u) - r(v)| + 2\varepsilon}. \tag{9}$$

Take $(\pi_\theta(a), \pi_\theta(b), \pi_\theta(c)) = (1, e^\theta, 1)/(2 + e^\theta)$ at $\theta = \theta_{\text{old}} = 0$. Then $\pi_0 = P$, and the log-probability derivatives $\partial_\theta \log \pi_\theta(u)|_{\theta=0}$ are $(-1/3, 2/3, -1/3)$. The clipped policy ratio is one at this point, so under the no-KL, no-entropy surrogate used in our experiments the one-token gradient contribution is $g(u, v) = A(u, v)\ \partial_\theta \log \pi_\theta(u)|_{\theta=0}$, holding the sampled advantage fixed.

Let $c_d = d/(\sqrt{2}d + 2\varepsilon)$ be the magnitude of the advantage for a reward gap $d \in \{1, 2\}$. Against a, responses b and c contribute $2c_1/9$ and $-c_2/9$ to $H(a)$; symmetry then gives

$$\big(H(a), H(b), H(c)\big) = \frac{2c_1 - c_2}{9}(1, 0, -1). \tag{10}$$

The dense comparison law P is uniform, so its positive and negative terms cancel. Under the direct-sparse law Q_s, the low-reward comparison a occurs more often than the high-reward comparison c, yielding

$$\begin{aligned} &G_{\text{dense}} = 0, \\ &G_{\text{corr}} = (Q_s(a) - Q_s(c))H(a) = \frac{2c_1 - c_2}{27} = \frac{\sqrt{2}}{27(\sqrt{2} + 2\varepsilon)(\sqrt{2} + \varepsilon)} > 0. \end{aligned} \tag{11}$$

Even though $W(b) = 1$, its expected contribution changes when the comparison response is drawn from the direct-sparse law Q_s rather than the dense law P. Thus exact response-wise weights do not generally recover the dense expected gradient. □

Here Q_s is the complete-response law of direct sparse rollout. In S^{3}R, q_s supplies token proposals, while the completed responses follow P, as proved next.

A.3 Dense response-group preservation under adaptive sparse proposals

Proof of Proposition 2. Fix a committed prefix $(x, o^i_{<t})$ and any drafter state determined by earlier sampling and verification. Let $p(v)$ be the dense verifier's probability of token v at this prefix, computed with dense historical KV, and let $q_s(v)$ be the proposal probability recorded when the candidate is sampled. Earlier acceptance decisions do not use later candidates, so conditioning on

their acceptance leaves the next candidate distributed according to its recorded q_s at the committed prefix.

The candidate is accepted with probability $\min\{1, p(v)/q_s(v)\}$; on rejection, the discarded suffix is replaced by a draw from the residual distribution in Eq. (2). The total rejection probability is $Z = \sum_u [p(u) - q_s(u)]_+$, so for $Z > 0$ the next committed token has probability

$$\min\{p(v), q_s(v)\} + Z r_s(v) = \min\{p(v), q_s(v)\} + [p(v) - q_s(v)]_+ = p(v). \tag{12}$$

If $Z = 0$, then $p = q_s$ and every candidate is accepted. The residual also covers tokens with $q_s(v) = 0$ when $p(v) > 0$.

The identity holds conditional on every preceding drafter state, including states selected using completed verification. Dense prefill and the bonus token after a fully accepted block also follow p; rejected suffixes are rolled back before subsequent verification. Thus every committed token has the dense conditional distribution, even as sparse proposals adapt. With the same termination rules, the chain rule gives

$$\Pr_{\mathrm{S}^3\mathrm{R},s}(o^i \mid x) = \prod_{t=1}^{|o^i|} \pi^{\mathrm{dense}}_{\theta_{\mathrm{old}}}(o^i_t \mid x, o^i_{<t}) = P(o^i \mid x). \tag{13}$$

All requested responses retain their group identities and use independent randomness. Their joint law is therefore $\prod_{i=1}^{G} P(o^i \mid x)$, as in Eq. (3). Applying the same reward, advantage, and training operations to these groups produces gradient estimates with the same distribution as dense AR rollouts. □

For direct sparse rollout, Eq. (1) gives the training ratio $\hat{w} = w(\theta) w_s$. For S^{3}R, Eq. (12) instead makes the committed token's behavior probability $p(v)$, so at $v = o^i_t$ the ratio is

$$\begin{aligned} \frac{\pi_\theta(v \mid x, o^i_{<t})}{\Pr_{\mathrm{S}^3\mathrm{R},s}(v \mid x, o^i_{<t})} &= \frac{\pi_\theta(v \mid x, o^i_{<t})}{\pi^{\mathrm{dense}}_{\theta_{\mathrm{old}}}(v \mid x, o^i_{<t})} \\ &= w(\theta). \end{aligned} \tag{14}$$

Thus the extra w_s factor in Eq. (1) does not enter the training update: q_s supplies proposals, while the committed tokens follow p.

B Implementation Details of S^{3}R

This appendix details the sparse drafting, full-attention verification, and supporting RL execution design in Section 4. Our rollout backend inherits the open-source implementation of Zhao et al. (2025) under veRL (Sheng et al., 2025) framework. We first specify sparse-context updates and verification state, then describe request scheduling, and finally connect their execution to the RL training loop in Algorithm 1.

B.1 Sparse-context construction and verification state

Budget and recent positions. For each layer ℓ, let $\mathcal{C}_\ell$ denote the KV indices retained after exact sampling: the pre-draft context and the accepted draft prefix; KV-head indices remain implicit. The implementation applies a minimum budget $K_{\min}$ and caps the ratio-based budget by the available context length. The number of positions retained at context refresh is

$$K_\ell = \min\big\{|\mathcal{C}_\ell|, \max\{K_{\min}, \lceil s|\mathcal{C}_\ell|\rceil\}\big\}. \tag{15}$$

If the preceding verification commits n output tokens, including its correction or bonus token, $\mathcal{R}_\ell$ contains the last $\min\{K_\ell, n+1\}$ valid cached positions around the verification boundary. The newly sampled correction or bonus token has no KV entry at this boundary; its KV is computed by subsequent execution. Each layer and KV head independently selects the remaining $K_\ell - |\mathcal{R}_\ell|$ positions from $\mathcal{C}_\ell \setminus \mathcal{R}_\ell$ using the verifier scores $\mathbf{a}_\ell$. This is Eq. (4) with K_ℓ replacing $\lceil s|\mathcal{C}_\ell|\rceil$. The selected historical positions remain fixed within a draft block, while $\mathcal{I}_\ell$ is extended with newly computed KV entries. Thus K_ℓ specifies the budget at refresh; the attended context grows with subsequent draft steps.

Attention operators. Each draft step applies attention to the selected KV entries:

$$\mathrm{SparseAttn}(\mathbf{q}; \mathcal{I}) = \mathrm{softmax}\left(\frac{\mathbf{q}\mathbf{K}_{\mathcal{I}}^{\top}}{\sqrt{d}}\right)\mathbf{V}_{\mathcal{I}}, \tag{16}$$

where $\mathbf{q} \in \mathbb{R}^{1\times d}$ is the current query and d the head dimension. For the verifier's query matrix $\mathbf{Q}$, full attention is

$$\mathrm{FullAttn}(\mathbf{Q}) = \mathrm{softmax}_{\mathrm{causal}}\left(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{d}}\right)\mathbf{V}, \tag{17}$$

where $\mathbf{K}$ and $\mathbf{V}$ cover the complete context and the proposed block.

Attention-score reduction. For each layer and KV head, $\mathbf{a}_\ell$ assigns each historical position the mean of its normalized attention weights over all query positions in the verified block and the query heads sharing that KV head. Queries for candidate tokens later rejected by exact sampling still contribute to this mean. A separate GPU kernel recomputes the necessary query–key scores using the verifier's saved softmax normalizers and stores only the reduced per-position scores. GPU Top-K consumes these scores; neither step materializes the full attention matrix or requires another Transformer forward.

Shared KV and proposal state. Sparse and full attention address the same KV storage; the sparse selection is an index view into the complete cache. Verification overwrites provisional draft KV entries with dense states. Once exact sampling determines the accepted draft prefix, KV entries for the unaccepted candidate suffix are discarded. For the current unverified block, each request also retains the full-vocabulary proposal distributions on the GPU. These distributions and the verifier's target distributions supply acceptance and residual sampling (Appendix A.3). The proposal buffer is reused after verification and released when the request completes.

B.2 KV-aware AIMD scheduler and mixed-stage execution

Concurrency update rule. At scheduler iteration j, the request-concurrency limit C_j determines whether additional requests can join the active batch of size B. Let u_j denote GPU KV-cache utilization and O_j the number of paused requests awaiting KV restoration from CPU. The backlog captures outstanding demand that GPU utilization alone misses after caches move to CPU. The KV-aware AIMD scheduler updates C_j using additive increase and multiplicative decrease:

$$C_{j+1} = \mathrm{clip}_{[C_{\min}, C_{\max}]}\left(\begin{cases} C_j + \delta, & u_j < u_{\mathrm{low}}, \\ \lfloor \beta C_j \rfloor, & u_j > u_{\mathrm{high}} \ \vee \ O_j > O_{\mathrm{th}}, \\ C_j, & \text{otherwise.} \end{cases}\right) \tag{18}$$

Here δ is a positive integer increment, $\beta \in (0,1)$ the reduction factor, and $C_{\min}, C_{\max}$ the concurrency bounds, with utilization thresholds $u_{\mathrm{low}} < u_{\mathrm{high}}$. The branches are evaluated in order: low utilization permits additive growth; otherwise, high utilization or an excessive backlog triggers multiplicative backoff. If B exceeds a reduced C_j, additional requests wait for B to fall below the new limit, while already active requests continue subject to KV capacity.

Request scheduling and KV capacity. ShadowKV combines low-rank key storage, value offloading, and sparse KV reconstruction to increase long-context inference throughput (Sun et al., 2025). Here, offloading preserves complete dense KV entries for subsequent full-attention verification. The KV-aware AIMD scheduler operates over a KV offload/restore mechanism. Each rollout replica maintains pending and active requests, plus a backlog of paused requests awaiting KV restoration from CPU. The concurrency limit C_j determines when additional requests can join the active batch, while a separate capacity check accounts for the KV storage required by the next execution batch. When resident KV state leaves insufficient space for execution, the scheduler prioritizes eligible requests with the smallest current KV-cache length, pauses them, and transfers KV pages to CPU memory. Token history, proposal state, and execution stage remain available for resumption. A paused request rejoins the active set after its KV state has been restored, subject to capacity and the concurrency limit. Restoring paused requests takes priority over starting new requests. Completed requests release their resources. The scheduler observes GPU KV utilization and the KV-restoration backlog and applies Eq. (18) to set the next iteration's concurrency limit. A reduction of this limit delays request activation while already active requests continue, subject to KV-capacity checks.

Mixed execution. We use mixed execution of prefill, draft, and verification requests. The active requests can occupy different stages. A prefill request contributes a prompt chunk, a drafting request contributes one query, and a verifying request contributes its candidate block and preceding committed token. These queries are packed into one model forward, with sparse attention for drafting and full attention for prefill and verification. Consequently, the active request count differs from the query count processed by the model. This execution implements the phase mixing analyzed in Appendix F.1. CPU scheduling overlaps GPU execution; each request starts its next draft block once verification and sparse-index updates are complete.

B.3 Execution in the RL training loop

At each RL step, the actor's parameters are synchronized to the rollout replicas, and this snapshot supplies both drafting and verification throughout generation. After complete response groups return to the trainer with their prompt and group identities, rollout resources are released for the actor update.

Resource lifetimes across RL steps. Table 3 tracks the transition from rollout generation to actor optimization and back. CUDA graphs use predefined batch-size buckets. After KV resources are released, captures tied to their storage are renewed for these buckets before the next rollout, allowing graph replay for the covered batch sizes.

Table 3: Resource lifetimes across the rollout and actor-update phases.

State	Before actor update	Before the next rollout
Requests and KV	Release request state and KV resources.	Start new requests with KV capacity fixed for the rollout.
Rollout model	Retain parameter buffers on GPU.	Synchronize the updated actor snapshot into these buffers.
CUDA graphs	Discard captures tied to released KV storage; retain the planned batch sizes.	Recapture the planned sizes for graph replay during generation.

Algorithm 1 combines this resource lifecycle with the per-request drafting and verification steps. Here $\mathcal{B}_j$ is the scheduled batch, m counts proposals in the current draft block, and o^i is the committed response prefix for request (x, i). The symbols $\mathcal{I}$, $\mathcal{R}$, and $\mathbf{a}$ denote the layer- and KV-head-specific state defined above. KVSchedule and MixedForward implement Appendix B.2; ExactSample returns the committed tokens under Appendix A.3.

C Detailed Experimental Setup

This appendix specifies the training, decoding, and measurement settings for Section 6, together with the baseline implementations and ablation configurations.

C.1 Infrastructure, Models, and Data

Infrastructure. We run the main training experiments on a single node with eight NVIDIA H100 GPUs, 64 CPU cores, and 1 TB of host memory. We use veRL (Sheng et al., 2025) v0.7.0 for training and vLLM (Kwon et al., 2023) v0.12.0 for the baseline rollout engines. Actor training uses Fully Sharded Data Parallel (FSDP). Rollout generation uses four replicas, each with tensor parallelism of two (TP2/DP4). All methods share the GPU allocation, KV-memory budget, and a maximum decoding batch size of 256 requests per replica. CUDA graphs, decoding warm-up, prefix caching, and chunked prefill are enabled for every method.

Models and training data. We initialize the target policies from Qwen3-1.7B and Qwen3-8B (Qwen Team, 2025), with thinking enabled and the default Qwen3 chat template. We use the original DAPO-MATH-17k dataset (Yu et al., 2025) without additional filtering and shuffle it with a fixed random seed. Answer extraction follows DeepScaleR (Luo et al., 2025). Responses that exceed the length limit, are truncated, or contain no valid extracted answer are scored as incorrect.

Algorithm 1 S^{3}R Pipeline

Input: Policy θ, training prompts
Parameters: group size G, draft length k, sparse budget $(s, K_{\min})$; KV-aware AIMD scheduler parameters in Eq. (18)

for each RL step **do**
  $\theta_{\text{old}} \leftarrow \theta$; synchronize rollout replicas
  Prepare fixed-capacity KV storage and CUDA graphs for planned batch sizes
  Initialize G requests per prompt; distribute across replicas
  for all rollout replicas in parallel **do**
    for scheduler iterations $j = 0, 1, \ldots$ until rollout completes **do**
      $\mathcal{B}_j \leftarrow \text{KVSCHEDULE}(C_j)$ ▷ Capacity, offload, restore, start requests
      $(q, p, \mathbf{a}) \leftarrow \text{MIXEDFORWARD}_{\theta_{\text{old}}}(\mathcal{B}_j; \mathcal{I})$ ▷ Eqs. (16)–(17)
      for all requests (x, i) in $\mathcal{B}_j$ **do**
        if prefill completes **then**
          $o_1^i \sim p; \quad o^i \leftarrow [o_1^i]$
          $\mathcal{I} \leftarrow \{1, \ldots, S\}; \quad m \leftarrow 0$
        else if a draft step completes **then**
          $m \leftarrow m + 1; \quad \tilde{o}_m^i \sim q_m$ ▷ Verify when $m = k$
        else if verification completes **then**
          $\Delta o^i \leftarrow \text{EXACTSAMPLE}(\tilde{o}_{1:k}^i, q_{1:k}, p_{1:k+1})$
          $o^i \leftarrow [o^i; \Delta o^i]$; discard unaccepted draft KV
          $S \leftarrow$ valid KV length; $\quad K \leftarrow \min\{S, \max\{K_{\min}, \lceil sS \rceil\}\}$
          $\mathcal{R} \leftarrow \{S - \min(K, |\Delta o^i| + 1) + 1, \ldots, S\}$
          $\mathcal{I} \leftarrow \mathcal{R} \cup \text{TopK}_{K-|\mathcal{R}|}(\mathbf{a}); \quad m \leftarrow 0$ ▷ Eq. (4)
        Collect completed responses; release their state
      Observe (u_j, O_j); update C_{j+1} by Eq. (18)
  Release request and KV resources for actor optimization
  Compute rewards and group-relative advantages from complete response groups
  Update θ using the policy objective and these advantages

C.2 Training and Method Configuration

RL training. We use the DAPO variant of GRPO with 256 prompts per training batch and eight responses per prompt, yielding 2,048 responses per step. Sampling and group filtering are disabled, so the generated batch is also the batch used for the actor update. The mini-batch size is 32, with dynamic batching enabled. We use AdamW with a learning rate of 10^{-5} and DAPO's asymmetric clipping parameters $(\epsilon_{\text{low}}, \epsilon_{\text{high}}) = (0.2, 0.28)$. The overlong penalty uses a 4K-token window. We use no KL loss, entropy bonus, or additional importance correction in the main comparison.

Sampling and evaluation. Training rollouts use temperature 1.0 without top-p or top-k filtering. The maximum response lengths are 16K tokens for Qwen3-1.7B and 24K tokens for Qwen3-8B. For evaluation, we use temperature 1.0 and top-$p = 0.7$. We report AIME26 and ArXiv-Math pass@1 scores from the best checkpoint, estimated using 16 sampled responses per problem as detailed in Appendix C.3.

S^{3}R. We run S^{3}R as the rollout backend in veRL (Appendix B.3). Both target models use a draft length of $k = 6$ and a sparse budget ratio of $s = 0.05$. The draft length counts only proposed tokens; the correction or bonus token is additional. The sparse KV budget is computed from the current context length, with a minimum budget of 192 tokens, following Appendix B.1. The KV-aware AIMD scheduler in Appendix B.2 uses $(C_{\min}, C_{\max}) = (160, 256)$, $\delta = 5$, $\beta = 0.7$, $(u_{\text{low}}, u_{\text{high}}) = (0.8, 0.95)$, and $O_{\text{th}} = 3$. Concurrency limits apply to each rollout replica.

C.3 Measurement and Reporting

Evaluation datasets. We use the MathArena releases (Dekoninck et al., 2026) of both evaluation benchmarks. AIME26 contains all 30 problems from the 2026 American Invitational Mathematics Examination, distributed as the `train` split of `MathArena/aime_2026`.[1] ArXiv-Math denotes

[1] AIME26: `https://huggingface.co/datasets/MathArena/aime_2026`.

the 103-problem ArXivMath collection covering December 2025 through March 2026, distributed as the `train` split of `MathArena/arxivmath`.[2] This collection contains 17, 23, 32, and 31 problems from December, January, February, and March, respectively. We evaluate on all problems in each benchmark.

Evaluation sampling and aggregation. For each evaluated checkpoint, we run one evaluation per benchmark and sample 16 responses per problem at temperature 1.0 and top-$p = 0.7$. This yields 480 responses on AIME26 and 1,648 on ArXiv-Math per checkpoint. Let N be the number of problems in a benchmark and c_i the number of correct responses among the 16 samples for problem i. The reported pass@1 score is

$$\widehat{\text{pass@1}} = \frac{100}{N} \sum_{i=1}^{N} \frac{c_i}{16} \quad (\%). \tag{19}$$

Thus, we average the sampled correctness rate for each problem across the benchmark, giving every problem equal weight. Each benchmark is aggregated separately.

Reporting window. The rollout time, training-step time, throughput, and block efficiency in Table 1 cover the full training run, excluding the first step for warm-up. All methods use the same reporting window within each model size.

Timing and throughput. Rollout time measures response generation. Training-step time measures the complete step, including reward evaluation, log-probability and advantage computation, actor updates, weight synchronization, and method-specific work such as historical-response cache updates. For each step, training throughput is $N_{\text{tok}}/(8T_{\text{step}})$ in tokens/s/GPU, where N_{tok} counts non-padding prompt and response tokens in the batch. We report the arithmetic mean of these per-step throughputs. Generation throughput in the KV-management ablation uses rollout time in the denominator with the same token and GPU accounting. Throughput speedups are ratios of the corresponding means relative to AR.

Scope of performance attribution. Table 1 evaluates the complete RL system against the specified baseline execution stacks. The lifecycle mechanisms in Appendix B.3 support repeated colocated execution, but their individual speed contributions are not isolated. Table 2 compares KV management and request scheduling within the sparse-speculative stack; it is not an ablation of the cross-step release and restoration paths.

Speculative decoding and KV diagnostics. Block efficiency τ is measured by directly counting the output tokens emitted per speculative round, including the correction or bonus token, and averaging over rounds in the reporting window. We also report draft-token acceptance rates. The KV-management ablation records peak CPU KV-cache occupancy, offload episodes, reoffloading, and transferred pages per 1,000 generated response tokens. Transfers include both device-to-host offloading and host-to-device reloading.

C.4 Baseline Implementations

AR. The AR baseline uses full-attention autoregressive decoding in vLLM with speculative decoding disabled. It is the timing and throughput reference for each target model.

N-gram Lookup. We use vLLM v0.12.0's prompt-lookup speculative decoding, following Saxena (2023). It matches the current token suffix against earlier occurrences in the prompt or generated response and proposes the following tokens. We retain the default lookup configuration, with both the minimum and maximum matching lengths set to five tokens.[3] Each speculative round proposes at most five draft tokens ($k = 5$). With this fixed proposal-length limit, positions after the first rejected token still incur target-model computation during parallel verification, although their outputs are

[2] ArXivMath: `https://huggingface.co/datasets/MathArena/arxivmath`.

[3] vLLM v0.12.0 speculative configuration: `https://docs.vllm.ai/en/v0.12.0/api/vllm/config/speculative/`.

discarded (Liu et al., 2026a). Low acceptance therefore limits cost amortization: the observed block efficiencies of $\tau = 1.31$ and 1.29 in Table 1 leave verification cost spread over only about 1.3 emitted tokens per round, limiting rollout gains despite inexpensive lookup-based proposals.

History Retrieval. We implement RhymeRL (He et al., 2025) using veRL's SpecRL recipe. A distributed suffix-tree cache stores historical responses by prompt and is updated asynchronously after rollout generation. Each request queries the cache with up to seven trailing tokens of its current sequence, using a minimum retrieval token probability of 0.1. The proposal length adapts to the number of output tokens emitted in the preceding verification round. Retrieved tokens are verified as deterministic proposals using rejection sampling with residual correction.

Auxiliary Proposer. This baseline represents the learned auxiliary drafting approach used in FastRL (Hu et al., 2026) and NeMo RL (Iso et al., 2026). We use the pretrained EAGLE3 (Li et al., 2025b) checkpoints `Qwen3-1.7B_eagle3` and `Qwen3-8B_eagle3` released by AngelSlim on Hugging Face.[4] Both use chain drafting with a fixed draft length of $k = 5$ in the veRL–vLLM stack. The auxiliary weights remain frozen throughout the main experiments as the target policy is updated. This comparison focuses on rollout acceleration without additional drafter training. Checkpoint portability and online co-training are examined separately in Appendix D.

C.5 Ablation Configurations

Direct sparse + mismatch correction. This baseline in Figure 4(a) removes speculative decoding from our sparse rollout system and applies Sparse-RL's correction methods. It uses the main experiment's model, data, optimizer, batching, and sparse-budget settings. It generates responses with the sparse policy and applies the corrections in Sections 4.2–4.3 of Luo et al. (2026): sequence-level masking for anomalously small dense-to-sparse probability ratios and tokenwise importance reweighting outside the clipped policy objective. This is the correction-based sparse-rollout variant discussed in Section 3.1 and Appendix A.1.

KV management and request scheduling. Table 2 uses ten RL steps after warm-up under the same training and decoding configuration as the main Qwen3-1.7B experiment. Recompute-256 restores evicted KV states by recomputation, whereas Offload-128, Offload-160, and Offload-256 use host-memory offloading with fixed per-replica concurrency limits of 128, 160, and 256. The KV-aware AIMD scheduler uses the main experiment's offloading mechanism and concurrency update rule with the parameters above. All four variants prioritize requests with the smallest current KV-cache length for transfer to CPU, following Appendix B.2.

D Checkpoint Portability and Online Drafter Adaptation

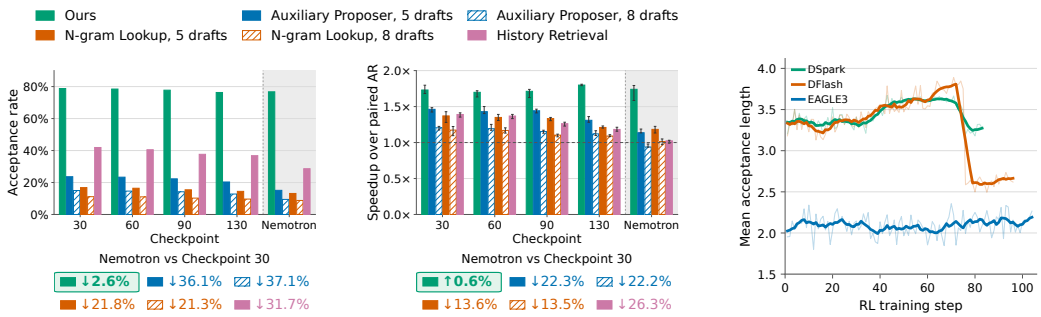
An RL workflow may start from an independently post-trained checkpoint for which a matched auxiliary proposer is unavailable. We examine two practical aspects of this setting: applying speculative decoding to different target checkpoints without additional drafter training, and maintaining an auxiliary proposer as the target policy evolves. These experiments complement the main training results by evaluating the flexibility of sparse self-drafting. Figure 7 compares checkpoint portability and periodic online drafter adaptation.

D.1 Checkpoint Portability without Auxiliary Proposer Training

Targets and setup. We evaluate five Qwen3-8B-family targets: steps 30, 60, 90 and 130 from corresponding checkpoints; and the released Nemotron-Cascade-8B model (Wang et al., 2025a). The latter has undergone both supervised fine-tuning and reinforcement learning. These targets are treated as separate categories: Nemotron do not extend the training trajectory of steps 30–130.

All targets use the same 30 AIME24 problems, with 16 responses per problem, temperature 0.6. This checkpoint study uses default S^3R as the setup in our main experiment. Auxiliary Proposer uses the

[4] `https://huggingface.co/AngelSlim/Qwen3-1.7B_eagle3` and `https://huggingface.co/AngelSlim/Qwen3-8B_eagle3`.



(a) Checkpoint acceptance. (b) Checkpoint speedup. (c) Online drafter adaptation.

Figure 7: Checkpoint portability and periodic online drafter adaptation. (a) Draft-token acceptance and (b) generation speedup relative to paired AR across target checkpoints; error bars summarize placement variability checked against additional runs. (c) Mean acceptance length during periodic co-training, with faint observations and bold seven-step rolling means.

fixed `AngelSlim/Qwen3-8B_eagle3` drafter. Auxiliary Proposer and N-gram Lookup propose five draft tokens, while History Retrieval uses adaptive suffix-cache drafting with a maximum of 16 tokens. All methods are evaluated without additional drafter fitting or online drafter updates. For this checkpoint study, History Retrieval's prebuilt checkpoint-0 trajectory cache is held fixed during target evaluation.

We exclude RL optimization time and are distinct from the main paper's end-to-end training speedups. Draft-token acceptance measures the fraction of proposed tokens accepted. The direction glyphs below each figure summarize each method's relative change from DAPO step 30 to Nemotron, computed as $(v_{\text{Nemotron}}/v_{\text{DAPO30}} - 1) \times 100\%$ for the corresponding metric v. These annotations compare separate target categories rather than a continuous training trajectory.

Results. Across the five targets, S^{3}R retains 76.32–78.78% draft-token acceptance and 1.698–1.799× generation speedup (Figure 7). Within the shared steps-30–90 trajectory, acceptance decreases modestly from 78.78% to 77.76%, while speedup remains at 1.698–1.731×. On Nemotron, S^{3}R achieves 76.74% acceptance and 1.740× speedup without fitting a new drafter. Fixed Auxiliary Proposer remains usable on this target: five drafts yield 1.136× speedup, whereas eight drafts yield 0.938×. Thus, direct reuse can provide reduced and configuration-sensitive benefits. The result supports the practical value of obtaining proposals directly from a selected supported checkpoint; it does not establish that retraining an auxiliary proposer is necessary for every target.

D.2 Periodic Online Drafter Adaptation

Online drafter setting and training. We analyze three Qwen3-8B GRPO baselines on DAPO Math in our VERL–SpeCo implementation, using EAGLE3, DFlash (Chen et al., 2026a), and the DSpark adaptation recipe (Cheng et al., 2026). Each run uses one node with 16 NPU workers, five rollouts per prompt, and prompt/response limits of 512/8,192 tokens. Target policies are optimized with AdamW at a learning rate of 10^{-6}. Target hidden states are collected through the old-logprob forward pass, and the auxiliary drafter is trained every five RL steps per update. EAGLE3 uses a prompt batch size of 64, TP$=$ 2, and three draft tokens; DFlash uses a batch size of 64, TP$=$ 2, and 15 draft tokens; DSpark uses a batch size of 64, TP2, and seven draft tokens. Thus, drafter optimization is periodic rather than accompanying every target-policy update. Increasing drafter optimization times will increase system overhead.

Observations and implications. Figure 7c shows the mean acceptance length τ for each run. We use this metric to describe acceptance changes within each run. In these configurations, periodic adaptation does not produce a monotonic increase in proposal acceptance. Comparing the same windows, steps 61–75 and 76–82, mean acceptance length decreases from 3.603 to 3.245 for DSpark and from 3.761 to 2.607 for DFlash, while EAGLE3 remains approximately stable, changing from

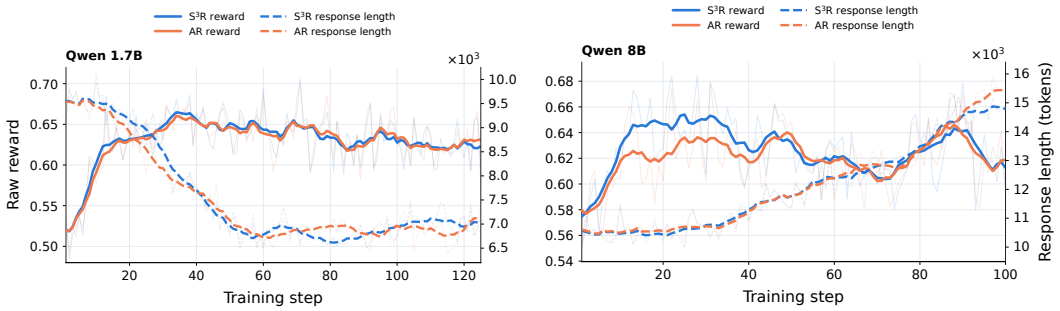
2.075 to 2.124. We assume that even co-training the drafter and the target policy requires specific adaptation to avoid drafter staleness problems.

Successful online co-training has sbeen reported in NeMo-RL (Wang et al., 2026b), which supports EAGLE3, DFlash, and DSpark through branch-aware context parallelism and cross-stage target-feature transport, accelerating RL while closely tracking the baseline learning trajectory. The present case concerns the periodic adaptation configuration used here. Maintaining acceptance can require coordinating auxiliary proposer adaptation with policy evolution. S^{3}R removes the need to select and initialize a separate drafter and schedule its optimization: its sparse proposals and dense verification share the synchronized rollout-policy snapshot. Acceptance can still vary with the sparse approximation and the target distribution, but this design avoids a separate source of parameter lag and auxiliary training effort.

E Additional Rollout Acceleration Diagnostics

We supplement the main results with reward and response-length trajectories for both model sizes, acceptance measurements across draft positions, and the Qwen3-1.7B throughput and timing results.

E.1 Training Dynamics across Model Sizes



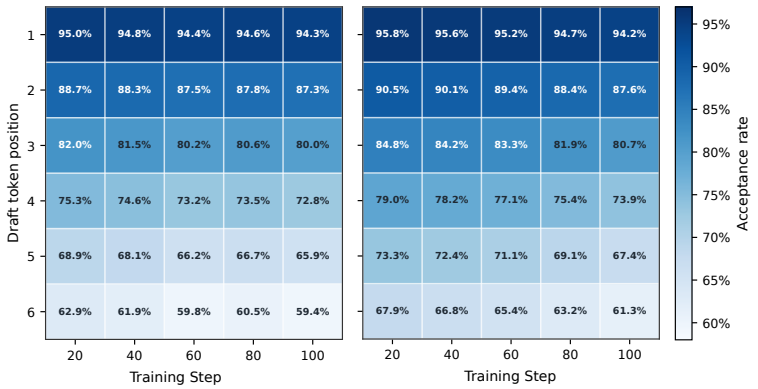
(a) Qwen3-1.7B. (b) Qwen3-8B.

Figure 8: Reward and response-length trajectories of AR and S^{3}R across training steps on Qwen3-1.7B and Qwen3-8B.

Figure 8 shows comparable reward and response-length trajectories for AR and S^{3}R across both model sizes. On Qwen3-1.7B, reward rises rapidly before leveling off, while responses become shorter. Qwen3-8B exhibits the opposite length trend, with responses growing throughout training; despite some early reward differences, the later reward trajectories remain close. S^{3}R follows AR in both cases, showing that its acceleration is compatible with similar learning behavior under both decreasing and increasing response lengths.

E.2 Acceptance across Draft Positions

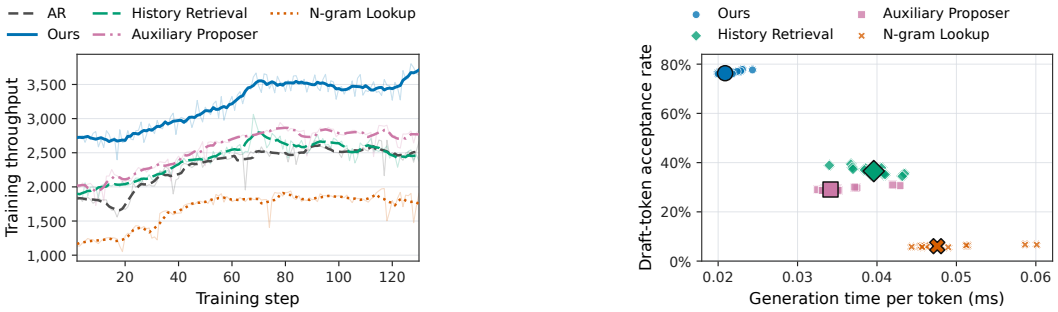
Draft-token acceptance is strongly position dependent (Figure 9). For Qwen3-8B at step 20, acceptance falls from 95.8% for the first proposed token to 67.9% for the sixth. From step 20 to step 100, first-position acceptance decreases by 1.6 percentage points, whereas sixth-position acceptance decreases by 6.6 points, and the checkpoint-level rate falls from 81.9% to 77.5%. Qwen3-1.7B exhibits the same ordering: acceptance at the first and sixth positions changes from 95.0% and 62.9% to 94.3% and 59.4%, respectively, corresponding to declines of 0.7 and 3.5 percentage points. Thus, later speculative positions have both lower acceptance and larger declines during training. It is highly related to the entropy dynamics, showing that the RL is encouraging exploration (Figure. 5c). This pattern motivates exploring an adaptive draft horizon that shortens when late-position acceptance deteriorates, which we left for future work.



(a) Qwen3-1.7B. (b) Qwen3-8B.

Figure 9: S^{3}R draft-token acceptance across training checkpoints for (a) Qwen3-1.7B and (b) Qwen3-8B. Cells report acceptance at each draft position.

E.3 Qwen3-1.7B Throughput and Timing–Acceptance Results



(a) Training throughput. (b) Timing–acceptance trade-off.

Figure 10: Qwen3-1.7B rollout diagnostics. (a) End-to-end training throughput; (b) generation time per processed token versus draft-token acceptance; outlined markers denote method medians.

Throughput. End-to-end throughput counts processed prompt and response tokens over the complete training-step time and GPU count; speedups are ratios of mean per-step throughput. Figure 10a shows the throughput trajectory during training. The mean throughput in Table 1 is 3,213.8 tokens/s/GPU for S^{3}R and 2,308.6 for AR, a $1.392\times$ ratio. Auxiliary Proposer, History Retrieval, and N-gram Lookup reach $1.110\times$, $1.045\times$, and $0.724\times$ the AR throughput, respectively.

Acceptance and generation cost. In Figure 10b, S^{3}R has median draft-token acceptance of 76.3% at 0.0209 ms per processed token, compared with 29.1% at 0.0341 ms for Auxiliary Proposer and 36.6% at 0.0396 ms for History Retrieval. Its median generation-time metric is 38.9% lower than Auxiliary Proposer's and 47.3% lower than History Retrieval's. Although History Retrieval has higher acceptance than Auxiliary Proposer, it also incurs greater time per processed token. This agrees with the 8B comparison: acceptance must be considered together with proposal, verification, and runtime costs.

F WHY SELF-SPARSE SPECULATIVE DECODING FOR RL ROLLOUTS?

Sparse self-drafting is useful when the attention work it removes outweighs its retained projection work and dense verification cost. This appendix supports Section 5 by connecting that cost balance to measured rollout workloads. We first derive the latency condition, then trace the evidence from attention savings to realized throughput, and use a simple speedup surface to explain the budget–acceptance trade-off.

F.1 LATENCY DECOMPOSITION FOR SELF-SPARSE SPECULATIVE DECODING

We extend the serialized latency accounting in Eq. 5 by separating projection work from historical KV reads at locally fixed active batch size B and mean cached context length S. For complete rounds, let α be the mean fraction of the k draft tokens accepted; then $\tau = 1 + k\alpha \in [1, k+1]$.

Let $T_{\mathrm{GEMM}}(B)$ cover QKVO, FFN, and LM-head work, and let $T_{\mathrm{Attn}}(M)$ cover attention at aggregate KV volume $M \propto B \cdot S$. The shared-weight drafter retains projection work but reads a fraction $s \in (0, 1]$ of the historical KV. For bandwidth-bound attention at fixed effective bandwidth, the Roofline model (Williams et al., 2009) motivates approximating latency as linear in M. Verification processes the k drafts plus the preceding round's final token, whose dense KV is not yet cached. Assuming $k + 1 \ll S$ and shared historical reads across these $(k + 1)B$ queries gives

$$\begin{aligned} T_p &= T_{\mathrm{GEMM}}(B) + T_{\mathrm{Attn}}(M), \\ T_q &= T_{\mathrm{GEMM}}(B) + sT_{\mathrm{Attn}}(M), \\ T_V &= T_{\mathrm{GEMM}}((k+1)B) + T_{\mathrm{Attn}}(M). \end{aligned} \tag{20}$$

Projection latency need not scale linearly with B because small batches reuse weights. Larger batches can increase T_V/T_p, whereas longer contexts at fixed B push it toward one (Sadhukhan et al., 2025; Kim et al., 2026).

Shared weights also permit draft and verification queries in one forward. With requests evenly staggered across k draft stages and one verification stage (Zhao et al., 2025, Sections 3.2–3.3), each iteration has $kB/(k+1)$ single-query drafts and $B/(k+1)$ verifications of $k+1$ queries each. This yields $(2k + 1)B/(k + 1)$ queries and $(ks + 1)M/(k + 1)$ effective KV reads per iteration. Each request completes a round in $k + 1$ iterations, so the mixed-stage cost per output token is

$$\frac{T_{\mathrm{mix}}}{\tau} = \frac{k+1}{\tau}\left[T_{\mathrm{GEMM}}\left(\frac{2k+1}{k+1}B\right) + T_{\mathrm{Attn}}\left(\frac{ks+1}{k+1}M\right)\right]. \tag{21}$$

By linearity, the attention term becomes $(ks + 1)T_{\mathrm{Attn}}(M)/\tau$.

At full acceptance, $\tau = k + 1$ maximizes output per round for fixed (B, S, s, k). Comparing Eq. 21 with the AR latency T_p gives the ideal mixed-stage feasibility condition

$$k(1 - s)T_{\mathrm{Attn}}(M) > (k + 1)\left[T_{\mathrm{GEMM}}\left(\frac{2k+1}{k+1}B\right) - T_{\mathrm{GEMM}}(B)\right]. \tag{22}$$

The left-hand side is the attention saving; the right-hand side is the projection overhead from mixing draft and verification queries. If the condition fails, no acceptance rate yields a speedup in this component model. Observed acceleration also depends on actual acceptance and costs beyond this approximation, examined in Appendix F.3.

F.2 FROM ATTENTION SAVINGS TO ROLLOUT SPEEDUP

The measurements below follow the cost balance in three steps: identify where attention dominates, account for the drafting and verification costs, and test whether the resulting savings improve output throughput.

Experimental setting. Table 4 summarizes the component-profiling setup. Here B is the active batch on one TP replica, S is context length per request, and $N_{\mathrm{KV}} = BS$ is the aggregate live KV volume, with $1\mathrm{K} = 1024$ tokens. The subsequent throughput sweep uses real mathematical continuations and stochastic sampling, as specified below.

Table 4: Settings for the component measurements in Tables 5–7 and Figure 11.

Item	Setting
Model	Qwen3-8B, DAPO step-60 checkpoint
Hardware	2×NVIDIA H100 80GB HBM3
Parallelism	one rollout-engine replica, TP=2
Precision	BF16
Workload	equal-length synthetic prompts, fixed concurrency, ignore EOS
Sampling	temperature 0 (deterministic systems workload)
AR baseline	ordinary AR with full attention
Sparse draft	Pillar attention, budget ratio 0.05, minimum 192 tokens
Speculation	draft length $k = 6$, exact full-model verification

F.2.1 Larger live KV volumes make attention the bottleneck

At small live KV volumes, sparse attention targets only a modest share of decoding cost. The first four rows of Table 5 use the rollout-tail coordinates studied by EfficientRollout (Kim et al., 2026). Attention accounts for 18.8–22.4% of the measured kernel time, while FFN, QKVO, and LM-head projections together account for 62.1–65.0%. As context grows at fixed $B = 16$, attention rises from 2.34 ms at $S = 4$K to 12.93 ms at $S = 32$K, while FFN stays near 2.24 ms. This transition explains why S^{3}R targets historical KV reads. EfficientRollout's quantized self-drafter reduces weight traffic in projection-dominated tails; sparse self-drafting addresses the growing attention cost at larger live KV volumes. The two motivations are consistent with different workload regimes, so the relevant drafting strategy depends on which cost dominates.

Table 5: Full-attention AR decoding costs: ms per iteration (share of GPU kernel time), excluding TP collectives. The upper rows cover small-KV rollout-tail workloads; the lower rows show the transition to attention-dominated decoding. QKVO denotes query, key, value, and output projections.

N_{KV}	S	B	Attention	FFN	QKVO	LM head	Other
8K	2K	4	0.96 (19.4%)	2.18 (43.8%)	0.83 (16.6%)	0.21 (4.1%)	0.80 (16.1%)
8K	4K	2	0.93 (18.8%)	2.18 (44.2%)	0.82 (16.6%)	0.20 (4.2%)	0.80 (16.2%)
8K	8K	1	0.98 (19.8%)	2.17 (43.7%)	0.82 (16.5%)	0.20 (4.1%)	0.79 (16.0%)
16K	2K	8	1.16 (22.4%)	2.19 (42.1%)	0.83 (16.0%)	0.21 (4.0%)	0.81 (15.6%)
64K	4K	16	2.34 (36.2%)	2.24 (34.6%)	0.85 (13.2%)	0.21 (3.3%)	0.82 (12.7%)
128K	8K	16	3.85 (48.4%)	2.23 (28.1%)	0.85 (10.7%)	0.21 (2.6%)	0.82 (10.3%)
256K	16K	16	6.85 (62.4%)	2.24 (20.4%)	0.85 (7.8%)	0.21 (1.9%)	0.82 (7.5%)
512K	32K	16	12.93 (75.8%)	2.24 (13.2%)	0.85 (5.0%)	0.21 (1.2%)	0.82 (4.8%)
512K	16K	32	12.98 (75.1%)	2.37 (13.7%)	0.87 (5.1%)	0.21 (1.2%)	0.84 (4.9%)
512K	8K	64	13.11 (76.2%)	2.23 (13.0%)	0.76 (4.4%)	0.21 (1.2%)	0.89 (5.2%)
512K	4K	128	13.07 (75.2%)	2.32 (13.4%)	0.80 (4.6%)	0.22 (1.3%)	0.97 (5.6%)
512K	2K	256	13.44 (72.3%)	2.73 (14.7%)	0.99 (5.4%)	0.24 (1.3%)	1.18 (6.3%)

Figure 11 extends this comparison across feasible batch sizes $B \in \{1, 4, 16, 32, 64, 128, 256\}$ and context lengths $S \in \{1, 2, 4, 8, 16, 24, 32\}$K. Attention-to-FFN latency ratios are similar across different factorizations of the same N_{KV}: about 0.43–0.45 at 8K, 0.97–1.12 at 64K, and 4.92–5.88 at 512K. Thus, in this profiled range, aggregate live KV volume is a useful first-order indicator of the attention bottleneck. With eight responses per prompt in our RLVR setup (Section 6.1), concurrency can make attention expensive even at moderate context lengths. Conversely, a shrinking active batch can reduce this opportunity despite long remaining responses. These profiles motivate tracking both batch and context when deciding where sparse speculation is useful within an RL rollout, which is left for future work.

F.2.2 Attention savings must cover drafting and verification

Table 6 measures attention time for sparse drafting plus dense verification at eight feasible (B, S) settings. With a draft budget of $s = 0.05$, the reduction grows from 38.3% at $N_{\mathrm{KV}} = 64$K to 63.2–63.7% at 512K. The savings persist with dense verification included, supporting S^{3}R's decision to confine sparsity to proposals while paying for faithful target sampling. They reduce attention work

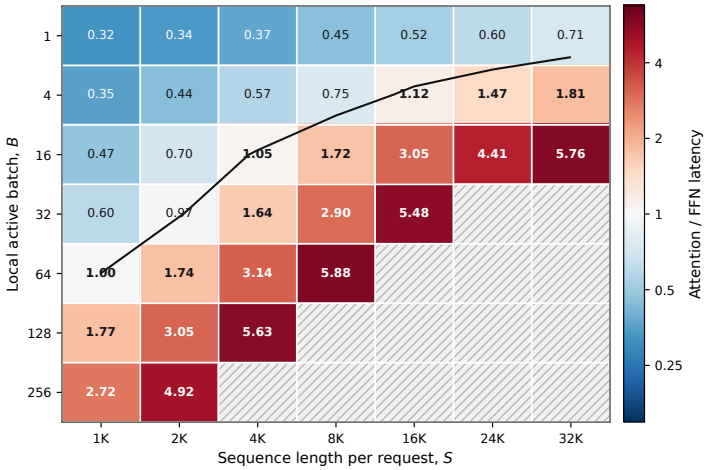


Figure 11: Attention-to-FFN kernel latency ratio for full-attention AR decoding across batch sizes and context lengths. The black contour marks equal latency; hatched cells exceed the measured KV-capacity limit.

without removing the complete KV state required for verification, which explains the need for the KV-aware runtime in Section 5. Whether these component savings translate into throughput depends on the retained projection work and accepted output.

Table 6: Attention time in AR and S^{3}R, in ms per scheduler iteration per TP rank (median over three runs). S^{3}R includes sparse drafting and dense verification, averaged over the full window, including idle iterations. These component savings must still be weighed against accepted output and other execution costs.

B	S	N_{KV}	Full attention	S^{3}R attention	Reduction
16	4K	64K	2.31	1.42	38.3%
16	8K	128K	3.83	1.89	50.6%
16	16K	256K	6.82	2.79	59.2%
32	4K	128K	3.80	1.96	48.5%
32	8K	256K	6.83	2.85	58.3%
32	16K	512K	12.91	4.68	63.7%
64	4K	256K	6.91	2.90	58.1%
64	8K	512K	13.02	4.79	63.2%

Table 7 adds these stage costs. Under the sequential accounting in Eq. 5, a round costs $6T_q + T_V$ and produces $1 + 6\alpha$ output tokens per request. Equating this cost to $(1 + 6\alpha)T_p$ gives the break-even acceptance

$$\alpha_{\mathrm{BE}} = \frac{1}{6}\left(\frac{6T_q + T_V}{T_p} - 1\right). \tag{23}$$

At 64K–128K live KV tokens, the measured stage costs exceed the AR reference even at full acceptance. At 256K, the threshold falls to 75.8–77.0%, and at 512K to 45.5–46.7%. The low-KV cases expose a limit of sharing policy weights: it avoids separate drafter updates, but retains projection work at every draft step. Where full acceptance is insufficient under this accounting, better proposal agreement alone cannot deliver acceleration. A future runtime could reduce stage costs or fall back to AR in such states, reserving sparse speculation for favorable workloads. These thresholds use separately timed sequential stages; complete-window timing below measures the realized benefit.

F.2.3 The sparse budget trades accepted progress against execution cost

To test this trade-off directly, we sweep the configured draft budget over 3%, 5%, 10%, 20%, 35%, and 50% for both Qwen3-1.7B and Qwen3-8B. We use frozen DAPO step-60 checkpoints, two H100 GPUs with TP$=$ 2, BF16, $k = 6$, and a minimum draft budget of 192 tokens. Each run

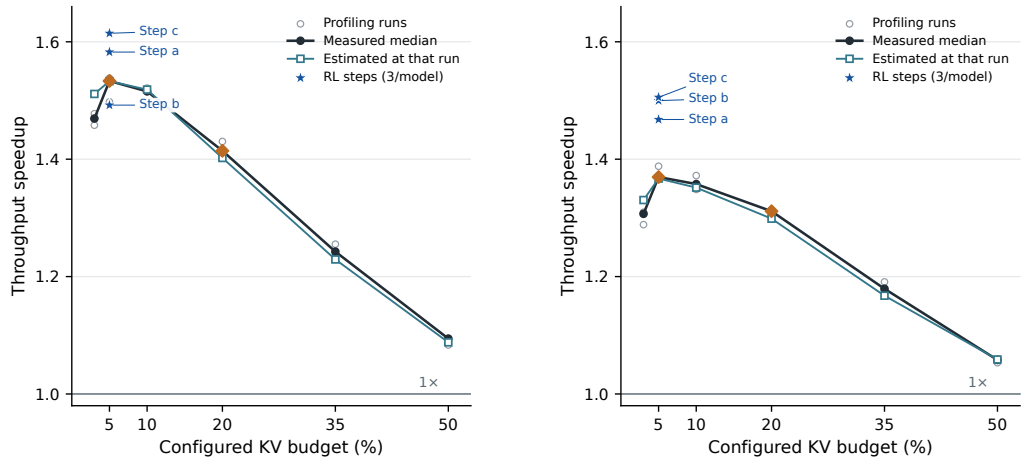
Table 7: Drafting and verification costs for $k = 6$ (ms, medians over three runs). The AR reference $7T_p$ generates as many tokens as a fully accepted round. α_{BE} is the acceptance needed to break even under sequential stage accounting. AR and drafting use CUDA Graphs; verification runs eagerly. The summed costs exclude idle iterations and cross-request phase mixing.

B	S	N_{KV}	AR ($7T_p$)	Draft ($6T_q$)	Verify (T_V)	Total	α_{BE}
16	4K	64K	54.42	40.06	28.06	68.12	$> 100\%$
16	8K	128K	65.55	39.15	27.97	67.12	$> 100\%$
16	16K	256K	88.51	42.38	28.10	70.48	76.2%
32	4K	128K	66.70	40.89	29.03	69.92	$> 100\%$
32	8K	256K	88.12	43.01	26.86	69.87	75.8%
32	16K	512K	133.18	43.73	28.56	72.29	46.7%
64	4K	256K	87.22	42.05	27.96	70.01	77.0%
64	8K	512K	129.85	42.51	26.68	69.19	45.5%

starts with $B = 64$ requests at $S = 8$K, using three fixed bundles of mathematical continuations sampled at temperature one. AR and S^{3}R use the same engine and inputs. Timing covers complete draft–verify rounds at fixed concurrency, with EOS disabled and no KV offloading. Speedup is the ratio of committed output tokens per second to the matched AR reference. This isolates the budget comparison from policy updates and request admission.

Figure 12 shows why acceptance alone cannot select the budget. Increasing the budget from 5% to 50% raises representative acceptance from 79.5% to 95.8% for 1.7B and from 80.4% to 96.7% for 8B, yet median speedup falls from $1.533\times$ to $1.094\times$ and from $1.370\times$ to $1.058\times$, respectively. Reducing the budget to 3% also lowers speedup, to $1.469\times$ and $1.307\times$. The 5–10% region provides a favorable balance in these workloads: further savings at smaller budgets lose useful progress, while larger budgets add more cost than their improved acceptance recovers. The sweep supports our default of 5%, without identifying a unique optimum.

Under Proposition 2, this throughput-driven budget choice preserves dense response groups. This is the practical value of S^{3}R's separation of proposal and behavior policies, given the instability of direct sparse rollout at the same 5% budget (Section 6.3). It motivates adapting the budget to accepted output per unit time as the policy and workload evolve.



(a) Qwen3-1.7B.

(b) Qwen3-8B.

Figure 12: Sparse-budget sweeps for (a) Qwen3-1.7B and (b) Qwen3-8B at $B = 64$, initial $S = 8$K, and $k = 6$. Black circles show median measured speedup; squares show estimates from Eq. 24 at the same runs' measured acceptance. Orange diamonds mark budgets excluded from fitting. Blue stars show separate RL observations, discussed in Appendix F.3.

F.3 Interpreting the budget–acceptance speedup surface

Figure 13 separates the budget sweep's two effects: drafting cost and accepted output. At fixed B, S, and k, write $B_{\text{mix}} = \frac{2k+1}{k+1}B$ for the query batch in a mixed-stage forward. Multiplying Eq. 21 by τ and using the linear-attention approximation $T_{\text{Attn}}(\lambda M) \simeq \lambda T_{\text{Attn}}(M)$ gives

$$T_{\text{mix}} \simeq (k+1)T_{\text{GEMM}}(B_{\text{mix}}) + (1+ks)T_{\text{Attn}}(M).$$

Only the draft attention term depends on s in this approximation. We normalize by the AR step cost in the same component model, $T_p = T_{\text{GEMM}}(B) + T_{\text{Attn}}(M)$, and add a dimensionless constant δ fitted to account for execution and approximation errors. Collecting the constant and budget-dependent terms yields

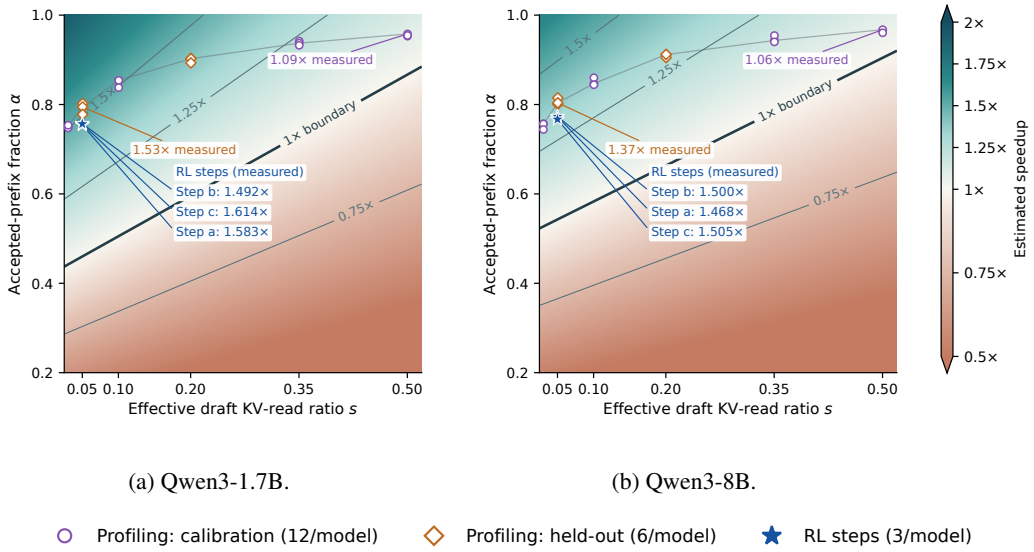
$$\frac{T_{\text{mix}}}{T_p} + \delta \simeq \underbrace{\frac{(k+1)T_{\text{GEMM}}(B_{\text{mix}}) + T_{\text{Attn}}(M)}{T_p} + \delta}_{c_0} + s\underbrace{\frac{kT_{\text{Attn}}(M)}{T_p}}_{c_1}.$$

Since each speculative round produces $\tau = 1 + k\alpha$ tokens per request, speedup is this output count divided by normalized round cost. Thus the calibrated estimate is

$$\widehat{\eta}(\alpha, s) = \frac{1+k\alpha}{c_0 + c_1 s}, \qquad k = 6. \tag{24}$$

Here c_0 includes projection work, dense verification attention, and the fitted correction; $c_1 s$ is the draft attention contribution. The ratio s is the effective fraction of historical KV tokens read by the drafter, accounting for the minimum budget, rounding, and newly appended tokens.

We profile component costs independently and fit only δ on the budget sweep, holding out the 5% and 20% budgets for validation. Acceptance enters as a measured input. Evaluating the expression over (s, α) gives the background in Figure 13, with measurements overlaid. Figure 12 applies the same model at each median-speedup run's acceptance and effective budget.



(a) Qwen3-1.7B. (b) Qwen3-8B.

Figure 13: Estimated speedup over sparse budget and acceptance for (a) Qwen3-1.7B and (b) Qwen3-8B at $B = 64$, $S \approx 8$K, and $k = 6$. The background evaluates Eq. 24; the dark contour marks $1\times$ speedup. Circles and diamonds show calibration and held-out observations. Blue stars locate separate RL observations at $s = 0.05$; their annotations report measured speedups.

Reading the surface. Moving upward at fixed budget increases accepted output and speedup; moving rightward at fixed acceptance increases cost. Along the measured trajectory, larger budgets improve acceptance, but their added cost eventually dominates. The dark contour marks where useful progress just covers round cost. All measured sweep points exceed $1\times$, so the break-even boundary and low-acceptance region remain model extrapolations. For S^3R's reuse of verification attention scores, the surface suggests a concrete objective: improve accepted progress at the same KV-read budget while keeping selection overhead small. Increasing context coverage alone need not move the system toward higher throughput.

Calibration and execution cost. The constant correction reduces held-out mean absolute percentage error from 84.61% to 0.80% for 1.7B and from 63.03% to 0.68% for 8B. This gap motivates measuring and optimizing verification, sparse selection, and scheduling alongside KV reads. It includes error in the component approximation, so the correction cannot identify these contributions individually. The small residual errors apply only to the profiled execution settings.

Connection to RL rollouts. Blue stars show three RL steps per model at $s = 0.05$. Pooling output counts and generation times across all four paired 1.7B steps and three 8B steps gives throughput speedups of $1.558\times$ and $1.491\times$, at acceptance 75.7% and 77.1%. These runs exclude optimization time and use changing batches, CPU KV offloading, and a vLLM AR baseline, so local fitting accuracy does not establish prediction accuracy for RL training. The complementary gain from KV-aware scheduling in Section 6.5 points to a further target: coordinate prompt sharing and prefetching with complete-group generation, so KV transfers do not consume the attention savings.