SPARROW: SPARSE ROLLOUT FOR STABLE AND EFFICIENT LONG-CONTEXT RL OF LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite being powerful, reinforcement learning with verifiable rewards (RLVR) induces extremely long CoT, thus making it highly computationally expensive. Since RLVR per-step training cost is dominated by long-context generation in rollout, sparse attention offers a promising way to accelerate dense rollout generation. However, using sparse rollouts in practice requires a delicate stability-efficiency tradeoff: overly aggressive sparsity leads to collapse, while overly lenient sparsity yields insufficient speedup. In this work, we study the optimal RL stability-efficiency tradeoff through the lens of sparse-to-dense actor-policy mismatch. We first observe that **sparse rollout collapse is not driven by uniform degradation across all tokens**: most tokens generated by the sparse actor align with the dense policy nearly perfectly, even under aggressive sparsity. Motivated by the observation, we hypothesize that sparse rollout training remains stable as long as the lower tail of the per-token acceptance rate is kept above a critical threshold throughout the sparse rollout trajectory. Then, we introduce a dynamic sparsity scheduling technique that maintains this tail statistic constant throughout generation and empirically validate our hypothesis. **Surprisingly, across a range of model sizes in Qwen3 thinking family, keeping the tail acceptance rate above a common empirical target (0.86) enables stable training.** We then use a cost-model analysis to find a sparsity schedule for maximum speedup under the mismatch threshold, achieving **2.2x**, **2.4x**, and **2.1x** rollout speedups when training Qwen3-1.7B, Qwen3-4B, and Qwen3-8B. Empirically, we show that the identified threshold generalizes to a much larger model (Qwen3-14B) and another RL domain (coding), enabling stable training. Additionally, our analysis naturally motivates DISTILLSPARSE. Through lightweight LoRA-based distillation directly on sparse rollouts, far more aggressive sparsity can attain the same acceptance-rate target, achieving higher speedup.

1 INTRODUCTION

Reinforcement Learning with Verifiable Rewards (RLVR) has recently driven state-of-the-art performance in mathematical reasoning and code generation (OpenAI et al., 2026; Guo et al., 2025), but its training cost is extremely high and dominated by long CoT rollout generation, over 70% of per-step cost (Wu et al., 2025). Agentic tasks (Anthropic, 2026; Team et al., 2025; Su et al., 2025) further exacerbate the bottleneck, as rollouts become even longer and can exceed the maximum context length. **Sparse attention** can effectively accelerate long-context generation (Tang et al., 2024a; Sun et al., 2024a; Yuan et al., 2025b) and thus can replace dense rollout to train the standard dense-attention policies. However, naively using sparse rollout for dense policies creates a **stability-efficiency tradeoff**: aggressive sparsity risks huge policy mismatch and training collapse, while mild sparsity brings RL stability with limited efficiency gains (Figure 1(a)). This motivates our central question: **how can sparse rollouts achieve maximal speedup while preserving the stability of dense-rollout RL training across model scales?**

Prior work falls short of addressing this problem. (1) **Sparse Attention Inference Accuracy** $\uparrow \neq$ **RL Stability** $\uparrow$: many sparse attention techniques have been developed to improve the efficiency–accuracy tradeoff for downstream-task inference (Tang et al., 2024b; Sun et al., 2024a; Xiao et al., 2024; Sadhukhan et al., 2025; Zhang et al., 2023). However, sparse rollout RL instability is mainly

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.

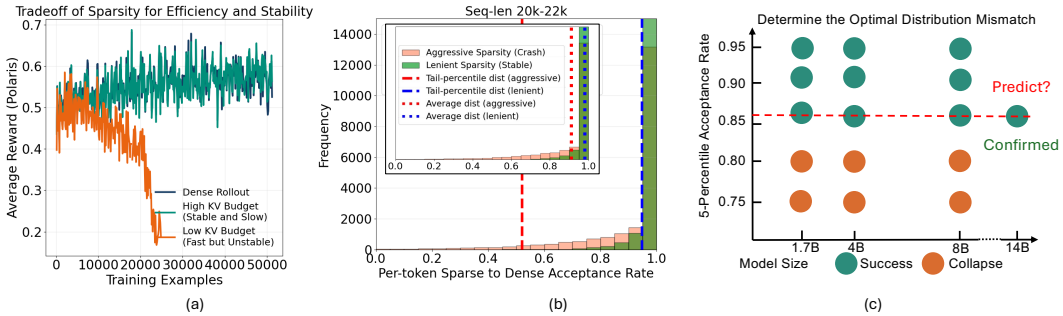

Figure 1: We study the tradeoff between training stability and efficiency using sparse rollout. (a) illustrates the main problem: arbitrary sparsity tends to miss the optimal tradeoff of lowest cost given stable training. (b) shows a case study demonstrating our insights, in the 20K–22K length bucket, regardless of lenient or aggressive sparsity, the per-token acceptance rate is long-tailed and heavily centered around 1. The high skewness of the distribution pushes its average acceptance rate toward 1, making it less informative to evaluate the distribution. Instead, we choose 5-percentile per-token acceptance rate statistics to focus on the tail distribution as the basis of our study. Panel (c) displays the main problem: what is the lowest distribution alignment needed to enable stable dense-policy training. For thinking-model sizes 1.7B, 4B, and 8B, we train each model with sparse rollout targeting various levels of distribution mismatch, thus identifying the lowest stable target among those tested. The threshold is consistent across model sizes within the family and is shown to generalize to 14B.

due to per-token distribution mismatch between sparse actor and dense policy rather than insufficient rollout rewards[1]. (2) **Suboptimal Convergence Under Severe Actor–Policy Mismatch**: prior work (Liu et al., 2025b;a; Luo et al., 2026) addresses actor–policy distribution mismatch, but often studies milder scenarios, such as staleness, where actor–policy KL divergence is one order of magnitude smaller than in sparse rollout. When directly applied, these techniques require clipping or masking significant training signals to maintain stability, leading to poor training convergence. (3) **Poor Efficiency**: Sparse rollout stability can be trivially recovered by applying elementwise Top-k or using a huge KV budget, but these approaches do not achieve large efficiency gains.

Ideally, we desire RL with sparse rollout to (1) enable stable dense-policy training, (2) match dense performance regardless of model size and generation length, and (3) achieve strong efficiency benefits over dense rollout.

Our study is heavily inspired by the following insight. The key observation is that **sparse rollout collapse is not driven by a uniform degradation across all tokens**. Even under aggressive sparsity, most generated tokens remain nearly distribution-aligned with the dense policy. The unstable signal instead appears in the small fraction of tokens where sparse and dense behavior diverge. As shown in Figure 1(b), we collect tokens from sparse-attention generations with 20K prompt length and 2K generation length, then measure each token's distribution mismatch against the dense model. The resulting distribution is highly skewed: most tokens are close to perfectly aligned, even when the sparsity budget is aggressive enough to cause RL collapse. (In fact, we show in Appendix M that the tail distribution can be well modeled by a Beta distribution.) **The high skewness makes average acceptance rate a weak stability indicator**. For example, when training Qwen3-1.7B with a 37K generation budget, a KV budget of 4096 remains stable while 2560 collapses, but their average per-token sparse-dense acceptance rates are still very close and very sensitive to measurement precision: 0.977 and 0.968. We therefore evaluate sparse-dense distribution mismatch with **lower-tail statistics** rather than the average. In particular, we use the lower 5-percentile per-token acceptance rate to measure the worst-aligned tokens while ignoring the many "perfect" tokens that are unlikely to cause RL training collapse. From the above formulation, we hypothesize the following.

***Hypothesis**: if the per-token tail acceptance rate between the sparse actor and dense policy stays above a threshold throughout rollout, sparse rollout for dense policy RL will be stable.*

[1] In Appendix A, we show a concrete ablation study that even if sufficient rollout rewards are provided, sparse to dense RL will still fail.

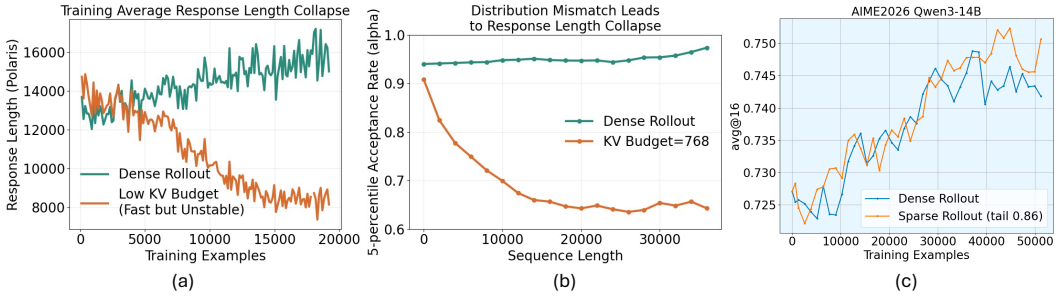

Figure 2: (a) Aggressive sparsity causes rollout response length to shrink during training, which correlates with dense-training collapse. (b) Unlike dense rollout, aggressive sparsity leads actor–policy distribution mismatch to grow sharply with sequence length. (c) We show that our hypothesis on the actor-policy mismatch target holds and is consistent across model parameter size. We generalize the threshold to 14B model and show that the training is stable across large number of training steps, on par in performance with dense rollout dense training.

However, there are **multiple challenges** in verifying this hypothesis. First, keeping per-token sparse-dense distribution mismatch constant across the entire trajectory is not straightforward. As shown in Figure 2(b), under any fixed sparsity budget, the per-token acceptance rate naturally deteriorates as the generation length increases. Second, after holding mismatch statistics constant throughout the trajectory, it remains difficult to know which threshold for the mismatch to target and how it scales as model size increases. Third, even if the above hypothesis holds, as shown in Figure 1(c), how to achieve the mismatch target with minimal inference cost remains non-trivial.

In Section 2, we systematically study sparse long-context rollout from the perspective of sparse-to-dense distribution mismatch and tame the RL instability when training dense policies to unleash the efficiency benefits from sparse-attention rollout. We use a range of model sizes from Qwen3 thinking family of models under extended RL training with a large and highly challenging math dataset, while enabling rollout length cutoff of 37K[2] to approximate real-life RL settings.

- In Section 2.2, we first introduce a technique called **dynamic sparsity scheduling**, which adjusts the sparsity budget to be more lenient as generation length increases, thus helping us maintain per-token tail acceptance rate statistics constant throughout rollout of 37K generation length.
- Using this technique, we conduct a systematic controlled study of the relationship between training stability and trajectory-level tail acceptance rate statistics. Across a range of models from the same Qwen3 thinking-model family, we search for a threshold of the tail statistics with each model that just keeps RL training stable, lower than which will crash. We validate our hypothesis and find that **surprisingly, the lowest stable target is consistent across model sizes in Qwen3 family**.
- Then, in Section 2.3, under this target, we formulate cost models to realistically and reliably compare and analyze dense and sparse rollout. We then use the cost model to derive a practical scheduling guideline for maximizing speedup under the identified per-token acceptance rate threshold for each model size.

Moreover, in Section 3, we put our analysis from the control study under extensive empirical validation. In Section 3.1, across the Qwen3 thinking family, keeping the tail acceptance rate level above a consistent threshold enables stable extended-step training, while dynamic sparsity scheduling achieves 2.2×, 2.4×, and 2.1× generation speedups for Qwen3-1.7B, Qwen3-4B, and Qwen3-8B, respectively. In Sections 3.2 and 3.3, we apply the same stability criterion to Qwen3-14B thinking-model RL and coding RL, where sparse rollout achieves performance on par with dense rollout. Following this analysis, in Section 3.5, we introduce DISTILLSPARSE to obtain substantial additional speedup on top of dynamic sparsity scheduling by enabling more aggressive sparsity while preserving the same stability criterion.

[2]The maximum model length for Qwen3 family models with 3K problem prompt length.

2 OBSERVATION AND INSIGHTS

Due to space limitations, we provide an extended discussion of prior techniques and our positioning of Sparrow in Appendix B. We study how sparse–dense mismatch predicts RL stability (Sections 2.1–2.2), then minimize rollout cost under the resulting stability constraint (Section 2.3).

2.1 PER-TOKEN ACCEPTANCE RATE: WHY LOWER-TAIL STATISTICS MATTER

At a fixed generated prefix, let P and Q be the dense-policy and sparse-actor next-token distributions over vocabulary $\mathcal{V}$. Their acceptance rate is the probability overlap (Leviathan et al., 2023):

$$\alpha(P, Q) = \mathbb{E}_{v\sim Q}\left[\min\left(\frac{P(v)}{Q(v)}, 1\right)\right] = \sum_{v\in\mathcal{V}} \min\{P(v), Q(v)\}. \tag{1}$$

Thus $\alpha \in [0, 1]$, with larger values indicating better alignment; $1 - \alpha = \mathrm{TV}(P, Q)$ measures mismatch. We compute per-token acceptance rate, comparing sparse actor to dense policy distributions at every token position. More details and discussions of top-k approximation are in Appendix G.

For a fixed KV budget sparsity, sparse–dense alignment deteriorates with sequence length; at a fixed length, smaller budgets reduce alignment. However, this degradation is concentrated in the lower tail, and **mean or per-token statistics can obscure it**, because most positions have acceptance rates above 0.999 regardless of position. For example, with Qwen3-1.7B Thinking and a KV budget of 512, the per-token acceptance rate has a Pearson correlation of only -0.1 with sequence length in the profiled range up to 40K tokens, even though the lower tail degrades markedly with length (Figure 3(a–c)). Figure 1(b) illustrates this concentration near one in the 20K–22K bin. **Budgets of 2560 and 4096 yield similar means (0.968 and 0.977), although the former causes training collapse and the latter does not**.[3]

Unlike the average acceptance rate, which is less informative when the distribution is highly skewed, statistics that focus on the tail distribution matter much more for evaluating the distribution mismatch between the approximate distribution and the original distribution. Smaller KV budgets produce a thicker lower tail of acceptance rates. In fact, we show that the tail distribution can be well modeled by a Beta distribution in Appendix M. In contrast, the 5th-percentile acceptance rate has a Pearson's r of -0.55 with sequence length, i.e., it is strongly correlated with length. As shown in Figure 4(a), the 5-percentile per-token acceptance rate allows the same sparsity configurations to show consistent mismatch values with the dense policy regardless of training iterations under the baseline dense rollout scenario: the values are almost the same between iteration 1 and iteration 550.

***Method**: We choose the 5-percentile cutoff of the per-token acceptance rate for our study to focus on the tail distribution of actor and policy mismatch in RL training.*

In Appendix H, we study N in the N-th percentile statistic and show that the choice of N does not matter. Under the same calibrated sparsity schedule, the 5th, 20th, and 30th percentiles map to one another consistently across generation length: a 5th-percentile value of 0.86 corresponds to roughly 0.95 at the 20th and 0.98 at the 30th percentile, and the ordering between schedules is preserved. We use the 5th percentile for convenience because it is numerically the most sensitive to mismatch.

2.2 CONTROL STUDY 1: WHAT THRESHOLD OF ACTOR–POLICY MISMATCH ENABLES STABLE RL TRAINING, AND HOW DOES IT SCALE WITH MODEL SIZE?

The main challenge in cleanly studying the effect of actor-policy mismatch on RL training stability with sparse attention is that when the sparsity budget is fixed before generation, the mismatch naturally increases as sequence length increases. We address it by introducing sparsity scheduling.

***Method**: We use **Sparsity Scheduling**, which increases the sparsity budget as the sequence length increases, thus preserving the tail distribution mismatch of every bin across the generation trajectory at a targeted value.*

[3]We estimate acceptance rate using the top-20 probabilities. As shown in Appendix G, due to the low output entropy of Qwen3 thinking models, the top-20 tokens cover over 99.9% of the probability mass.

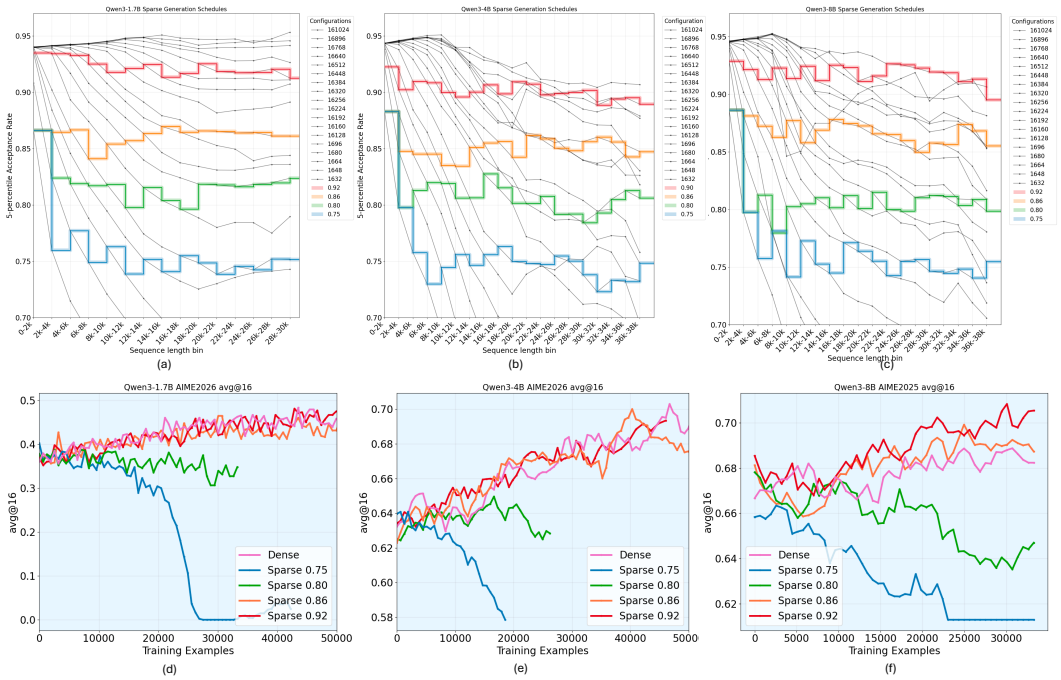

Figure 3: Control Study 1. Columns correspond to Qwen3-1.7B, Qwen3-4B, and Qwen3-8B. The first row plots the 5th-percentile acceptance rate versus sequence length for different sparsity configurations. We calibrate schedules to target bin-level acceptance-rate quantiles of 0.75, 0.80, 0.86, and 0.92. The second row tests whether each schedule enables stable training with performance on par with dense rollout.

For each Qwen3 model (1.7B, 4B, and 8B), we profile 18 KV budgets with page size 16 and varying page counts across 20 length bins (Figure 3(a–c)). We select a configuration in each bin to target acceptance rates (actor-policy distribution alignment level) of 0.75, 0.80, 0.86, or 0.92.

The calibration corpus used to compute acceptance rates consists of trajectories generated from AIME 2024–2026 prompts, each repeated eight times. We generate sparse trajectories and recompute dense log-probabilities to estimate per-token acceptance rates. Tokens are grouped into 2K-position bins; the 37K training cutoff spans 20 bins. Calibration details are in Appendix K. We then train dense policies using each calibrated schedule to test stability (Figure 3(d–f)). Results are surprising.

***Takeaway 1**: Across the Qwen3 thinking model family, the critical tail acceptance-rate threshold (τ) is around 0.86 (between 0.80 and 0.86), and τ is consistent across the model sizes we tested.*

Next, we minimize rollout cost under this empirical stability threshold of 0.86 for these models.

2.3 Control Study 2: What Is the Lowest Cost Needed for Stable and Performant Dense-Model Training?

To make sure the cost analysis can transfer to different hardware easily, we follow Sadhukhan et al. (2025) and model the cost of rollout based on model sizes and hardware memory bandwidth. For dense attention, we can formulate the compute, memory, and combined cost of repeated sampling N times in eFLOPs (equivalent FLOPS), given model parameters *P*, input prompt length in tokens L_{in}, output length in tokens L_{out}, dimension of both Key and Value *D*, GQA ratio (greater than 1) in *r*, and the arithmetic intensity of the hardware *I* (peak FLOP/s divided by memory bandwidth)

$$C_{comp} = 2 \cdot P \cdot N \cdot L_{out} + r \cdot (2 \cdot L_{in} + L_{out}) \cdot L_{out} \cdot N \cdot D$$

$$C_{mem} = 2 \cdot L_{in} \cdot L_{out} \cdot D + N \cdot L_{out}^2 \cdot D$$

$$C_{dense} = C_{comp} + I \cdot C_{mem} \tag{2}$$

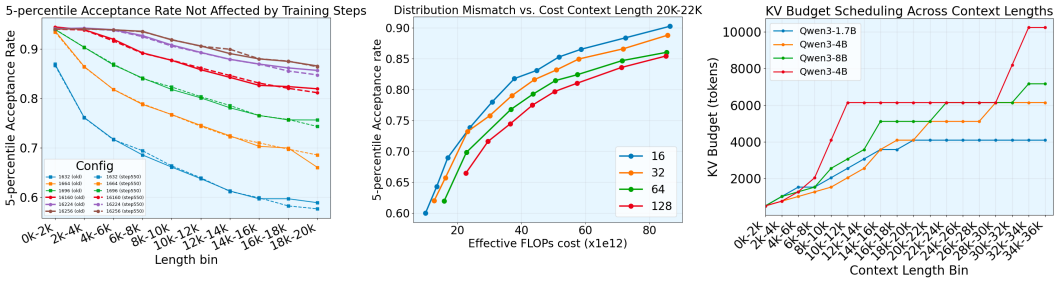

Figure 4: (a) shows an important property of the 5-percentile acceptance rate: it is not dependent on training iteration, and is instead determined by the dense model, sparsity level, and sequence length. (b) shows that page size 16 leads over larger page sizes in the tradeoff between distribution alignment and efficiency. (c) shows that larger models generally favor larger KV budgets to maintain the same distribution-alignment level (0.86 here).

Similarly, we can model the same for block-sparse attention as follows. For simplicity, we assume that Top-k kernels incur only minimal overhead for `pagesize` ≥ 16. Sparsity is with KV budget B and *pagesize*,

$$C_{\text{sparse, no scoring}} = \underbrace{2 \cdot N \cdot P \cdot L_{out} + 2 \cdot r \cdot N \cdot D \cdot B \cdot L_{out}}_{\text{compute}} + \underbrace{2 \cdot I \cdot N \cdot D \cdot B \cdot L_{out}}_{\text{memory}}.$$

$$C_{scoring} = \underbrace{2 \cdot N \cdot L_{in} \cdot D \cdot L_{out} + \frac{r \cdot N \cdot D \cdot L_{out}^2}{2 \cdot \text{pagesize}}}_{\text{compute}} + \underbrace{2 \cdot I \cdot L_{in} \cdot D \cdot L_{out} + \frac{I \cdot N \cdot D \cdot L_{out}^2}{2 \cdot \text{pagesize}}}_{\text{memory}}.$$

$$C_{sparse} = C_{\text{sparse, no scoring}} + C_{scoring} \tag{3}$$

We compare page sizes 16, 32, 64, and 128 for the 1.7B, 4B, and 8B models.[4] Page size 16 dominates the tail-acceptance–cost Pareto frontier: landmark scoring and top-k selection cost much less than attention computation. Figure 4(b) shows the 1.7B results at 20K–22K tokens; more visualizations and length regimes are in Appendix C.

> ***Takeaway 2**: For block-sparse attention (`pagesize` $\geq$ 16), more fine-grained sparsity (`pagesize` $\downarrow$) consistently beats the coarser sparsity (`pagesize` $\uparrow$) in distribution mismatch with the dense model across all cost regimes.*

Empirically, we found that if the top-k kernels are well implemented, the generation costs are also insensitive to page size, consistent with our cost model findings (Appendix C, Table 4). We therefore use page size 16 throughout the remaining experiments.

3 Empirical Studies

In this section, we evaluate sparse rollout across five empirical axes. Section 3.1 shows that dynamic sparsity scheduling maintains dense-level RL performance while achieving practical rollout speedups. Section 3.2 tests generalization: whether the same threshold transfers to Qwen3-14B, a model too large for exhaustive sparsity sweeps. Similarly, Section 3.3 shows that our findings generalize to coding RL. Section 3.4 shows that dynamic sparsity can be combined with other efficient rollout methods such as KV-cache quantization. Finally, Section 3.5 presents DistillSparse, which is naturally motivated by our analysis, enabling more aggressive sparse rollout with additional speedup.

3.1 Sparse Rollout Maintains RL Training Stability while Achieving Practical Speedup

Following the plan laid out in Section 2.2, we train Qwen3 (Yang et al., 2025) family models 1.7B, 4B, 8B for extended steps on Polaris (An et al., 2025) using the planned sparsity schedule based on the

[4]Our inference engine does not yet support page sizes below 16, which require further system optimization.

Table 1: Comprehensive Evaluation of Sparse Rollout relative to Dense, where we report the best-achieved scores under each downstream task and task-related metrics.

Models	AIME25 Mean@16	AIME25 Pass@16	AIME26 Mean@16	AIME26 Pass@16	AIME24 Mean@16	AIME24 Pass@16	AMC22/23 Mean@4	AMC12 Mean@4
Qwen3-1.7B Thinking 37K generation length cutoff; one epoch on 53K Polaris examples;								
Dense Rollout Dense Training	0.4322	0.5440	0.4833	0.5995	0.5687	0.7172	0.8072	0.7333
Sparsity setup (0.92) Rollout	0.4625	0.5729	0.4875	0.6197	0.5770	0.7155	0.834	0.7222
Sparsity setup (0.86) Rollout	0.4145	0.5436	0.4645	0.5914	0.5416	0.6958	0.8192	0.7166
Sparse with LoRA Distillation Sec. 3.5	0.4354	0.5562	0.4562	0.5874	0.5333	0.6929	0.8072	0.7333
Qwen3-4B Thinking 37K generation length cutoff; one epoch on 53K Polaris examples;								
Dense Rollout Dense Training	0.7145	0.7933	0.7229	0.8216	0.775	0.836	0.8	0.9367
Sparsity setup (0.92) Rollout	0.7	0.7978	0.7062	0.8158	0.7729	0.836	0.7944	0.9337
Sparsity setup (0.86) Rollout	0.7416	0.8321	0.7125	0.8197	0.756	0.832	0.7944	0.9397
Qwen3-8B Thinking 37K generation length cutoff; one epoch on 33K Polaris examples;								
Dense Rollout Dense Training	0.7083	0.8000	0.6979	0.8176	0.7770	0.8637	0.8111	0.9337
Sparsity setup (0.92) Rollout	0.7229	0.8076	0.725	0.8233	0.7815	0.8456	0.8111	0.927
Sparsity setup (0.86) Rollout	0.7145	0.8055	0.6979	0.8125	0.7791	0.8441	0.8277	0.9216
Qwen3-14B Thinking 37K generation length cutoff; one epoch on 53K Polaris examples;								
Dense Rollout Dense Training	0.75	0.8269	0.7770	0.8646	0.8125	0.8810	0.8055	0.9396
Sparsity setup (0.86) Rollout	0.75	0.828	0.7708	0.8558	0.8028	0.8696	0.8111	0.9367

tail acceptance rate statistics (5-percentile acceptance rate). Specifically, to enable high infrastructure utilization during training, we implement the dynamic sparsity scheduling generation required by our design based on open-sourced sparse attention library Vortex (Infini-AI-Lab, 2026).

The results are summarized in Table 1 with tests of three different years 2024, 2025, 2026 of AIME problems, each averaged over 16 samples, and AMC (2022 and 2023) and AMC12 (2024) problems, each averaged over 4 samples, showing that they are on par with dense.

Table 2: Throughput and speedup from dynamic sparsity scheduling, benchmarked on 4×H200 GPUs.

Model	Dense t.↑ (tokens/s)	Sparse t.↑ (tokens/s)	Rollout Speedup	RL One-step Speedup
Qwen3-1.7B (0.86)	2561	5662	2.21×	1.97×
Qwen3-4B (0.86)	1369	3308	2.41×	2.11×
Qwen3-8B (0.86)	1300	2682	2.06×	1.81×
Qwen3-14B (0.86)	1544	2289	1.49×	1.35×

Implementation Efficiency - To mimic the distribution of rollout during RL training of Polaris, we gather AIME24 and repeat every problem 8 times with 32K generation length cutoff. Results are shown in Table 2. For profiling, we deploy 4×H200 GPUs. We show that across 1.7B, 4B, and 8B, we can achieve over 2.0x speedup on RL rollout. Appendix I compares FP8 KV-cache quantization with dynamic sparsity and evaluates their composition and compatibility with asynchronous RL.

3.2 Testing Threshold Generalization on Much Larger Model

Beyond the three model sizes for which we have enough compute to sweep sparsity settings, we explore ways to extend our study to regimes where enumeration is no longer feasible. A Qwen3-14B Thinking model with a full 37K generation cutoff takes 4 H200 nodes (32 H200s) and 8 full days (∼190 hours) to train for an entire epoch. For us, it is no longer possible to brute-force all potential options for the optimal sparsity schedule that maximizes the efficiency benefit while preserving training convergence.

We follow the 5-percentile acceptance rate guidance to keep 0.86 and train Qwen3-14B using the determined sparsity schedule. The training runs for one epoch on Polaris. The dense policy from sparse rollout matches the dense policy from dense rollout RL very closely. We show the average training reward figure in Figure 2(c), and we show that the average training reward is highly close to dense reward with gap consistently less than 1%. We also populate the downstream evals in Table 1, showing that the Qwen3-14B matches the dense rollout training convergence. Appendix J reports full-epoch Qwen3-14B results and transfer to Llama.

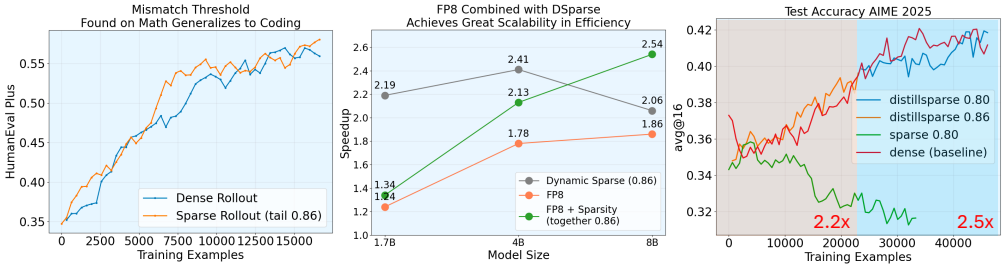

Figure 5: (a) We show the dynamic sparsity scheduling above the critical threshold can generalize to coding. We found that the 0.86 setting dynamic sparsity produces stable training on TACO datasets, matching performance of dense rollout dense training. (b) Dynamic Sparsity is orthogonal to other efficient rollout methods, and our threshold provides guidance to safely combine these techniques and achieve even better scalability in efficiency as the model size increases. (c) DISTILLSPARSE provides an opportunity to maintain training stability while gaining additional speedup.

Different Model Architecture Generalization - We show that there exists a tail acceptance-rate threshold for Llama-3 family of models also. In Appendix J.3, we show that dynamic sparsity can lead to stable RL training empirically for Llama-3.2-1B.

3.3 TESTING THRESHOLD GENERALIZATION ON CODING RL

Beyond math reasoning, training strong coding models also requires RL to scale up CoT to achieve higher pass-rate in harder coding problems. We show that the threshold found using math domain can be directly used for coding training. For the experimental implementation, we follow Liu & Zhang (2025) to set up a sandbox for reward computation and use the 21K filtered TACO training examples with 24K generation length cutoff from Li et al. (2023a) to train a Qwen3-1.7B Thinking model with the 0.86 sparsity setup. Following the same setting, we further train a Qwen3-4B Thinking model (with 32K generation length cutoff) with the 0.86 sparsity setup. We evaluate the training performance with LiveCodeBench (Han et al., 2024), HumanEval+, and MBPP+ (Liu et al., 2023). The results are reported in Table 3, where we show that the threshold determined in math reasoning generalizes to another generation workload, coding. We also present RL training convergence in Figure 5(a), showing that sparse rollout matches dense-rollout convergence over extended training.

Table 3: Coding RL evaluation for Qwen3-1.7B and Qwen3-4B trained on TACO.

Setups	LiveCodeBench	HumanEval+	MBPP+
Dense Rollout Baseline (1.7B)	0.3970	0.6219	0.5846
Sparsity (0.86)	0.3933	0.6280	0.6084
Dense Rollout (4B)	0.5542	0.7228	0.7271
Sparsity (0.86)	0.5551	0.7268	0.7301

Training Reproducibility - In Appendix H.2, we train the same dynamic sparsity schedule with 3 seeds and show that all runs are stable without significant performance variability.

3.4 DYNAMIC SPARSITY CAN COMBINE WITH OTHER EFFICIENT ROLLOUT METHODS AND LEAD TO BETTER SCALABILITY OF SPEEDUP

We here show that Dynamic Sparsity is orthogonal to other efficient rollout methods, and combination leads to better efficiency scalability when model size increases. Importantly, the identified tail acceptance-rate threshold can serve as a guidance to safely combine these techniques, leading to greater speedup while maintaining the training stability.

Traditionally, KV quantization can also lead to long-context generation speedup. Across 1.7B, 4B, and 8B in the Qwen3 family, we found that KV FP8 actor can consistently exceed the 0.86 tail acceptance-rate threshold, but KV FP4 cannot. We further discover that 1.7B KV FP8 combined with DSparse 0.93, 4B KV FP8 combined with DSparse 0.88, 8B KV FP8 combined with DSparse 0.88 can safely pass the 0.86 threshold. Specifically, we trained the rollout with the combined KV FP8 with DSparse 0.93 with dense policy and show that it is stable; due to space limits, details are shown

in Appendix I. We benchmarked the above three combined techniques across model sizes in Figure 5(b) and show that as the model size increases, **the speedup of the combined technique eventually surpasses dynamic sparsity alone**. The reason for the lower scalability of sparsity alone can be seen in Figure 4(c), as a higher budget is required to maintain the threshold as model size increases.

3.5 Case Study DistillSparse: Achieving the Same Stability Threshold with Lower Sparse-Rollout Cost

Since sparse actor to dense policy mismatch is the key to RL training stability, we discover that we can actively train the sparse actor with LoRA to shrink its mismatch to dense distribution, leading to even higher speedup from more aggressive sparsity: a technique we call DistillSparse. At each RL iteration, we generate trajectories with the sparse policy and reuse them to update the dense base model. We then freeze the base weights and update the sparse-policy LoRA adapter using the sparse PPO and on-policy distillation loss (Agarwal et al., 2024), before refreshing the rollout model. The adapter is active only for sparse rollout and on-policy distillation, which is an extra stage separated from dense policy training. The detailed algorithm and implementation are in Algorithm 1 of Appendix N.

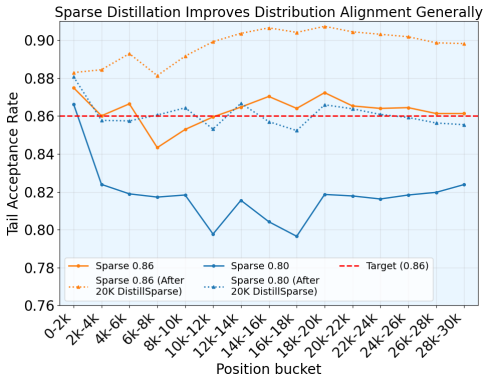

Figure 6: DistillSparse training improves sparse attention to dense distribution generally, beyond 0.86 sparsity. In fact, old sparsity with 0.80 tail acceptance rate now gets boosted to 0.86 close to the identified threshold.

Here we present our proof-of-concept case study: we run the experiment with Qwen3-1.7B Thinking model. With the goal of achieving training stability, we start with the 0.86 dynamic sparsity schedule (speedup 2.2x) but with DistillSparse LoRA activated. For every iteration trained, DistillSparse further boosts the distribution alignment of the 0.86 sparsity with dense.

We visualize the better alignment in Figure 6: after training for the first 23K training examples, the coral solid (before) and dashed (after) curves show the effect of added LoRA weights and on-policy distillation on the sparse rollout on the original 0.86 sparsity setting. Furthermore, we see that through distillation, the same sparsity level achieves significantly higher distribution alignment across all generation sequence lengths. Surprisingly, after DistillSparse, the original sparsity level of 0.80 (solid blue curve) now achieves a tail acceptance rate around the 0.86 threshold (red; dashed blue curve), and we switch to the 0.80 sparsity from then on. We show in Figure 5(c) that even with the switching, the training for another extended 23K training examples (46K training examples in total) is still stable and on par with dense for an extended number of steps, with full empirical validation at Table 1. For the second 23K training examples, we can train for 2.5x speedup in the rollout, higher than in the first 23K training examples. The gain of efficiency is caused by DistillSparse, as Figure 3(d) shows that training with 0.80 directly collapses the RL, and the DistillSparse training stability **further confirms the identified 0.86 tail acceptance-rate threshold for the model**. **Low Overhead** - Even as a proof-of-concept, we show that DistillSparse can be readily integrated into dynamic sparsity scheduling implementation with minimal extra overhead in Appendix D.

4 Conclusion

Sparse rollout can accelerate long-chain-of-thought RL, but distribution mismatch can destabilize dense-policy training. We identify lower-tail acceptance rates as a practical stability indicator and use them to calibrate dynamic sparsity schedules that preserve training performance across the evaluated model scales and tasks. DistillSparse further improves alignment with minimal overhead, enabling more aggressive sparsity and faster rollouts.

AI USE STATEMENT

We used AI tools to accelerate Triton kernel development for the implementations evaluated in the speedup table and for efficient DistillSparse execution. The authors independently developed the research plan and observations and conducted the experiments. We also used AI to improve paragraph concision and layout, reducing an approximately 11-page main-text draft written by the authors to nine pages. The manuscript was therefore based on an author-written draft rather than generated entirely by AI. The authors reviewed and proofread the text and code and take responsibility for the final text, code, and all other artifacts.

We used generative AI tools to assist with writing. We did not use generative AI tools for any other tasks that require disclosure. Additionally, we used generative AI tools to help create and edit software code and to edit the paper to improve readability. We have reviewed all AI-assisted work; for example, we verified the generated code for correctness and manually reviewed every LLM-generated paper modification. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Reducing the computational cost of RL training can make research on language-model reasoning more accessible, particularly to groups with limited resources. However, cheaper post-training may also accelerate the development of models that can be misused for misinformation, spam, or unsafe automated assistance. The societal impact of this systems contribution therefore depends on downstream evaluation and deployment. We encourage practitioners to accompany efficiency improvements with capability assessments, application-specific safeguards, and responsible release practices, as discussed further in the Broader Social Impact section.

REPRODUCIBILITY STATEMENT

We support reproducibility by describing the sparsity calibration and cost model in Section 2, and the training and evaluation setup in Section 3. The appendices provide cost-model implementation details (Appendix C), calibration guidance (Appendix K), repeated-seed results (Appendix H), and the DistillSparse configuration and algorithm (Appendices L and N). Upon publication, we will release the project code on GitHub to facilitate reproduction of our results and further research.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024.

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*, 2024.

Chenxin An, Zhihui Xie, Xiaonan Li, Lei Li, Jun Zhang, Shansan Gong, Ming Zhong, Jingjing Xu, Xipeng Qiu, Mingxuan Wang, and Lingpeng Kong. Polaris: A post-training recipe for scaling reinforcement learning on advanced reasoning models, 2025. URL https://hkunlp.github.io/blog/2025/Polaris.

Anthropic. Claude code. https://code.claude.com/, 2026. AI coding assistant.

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

Zhuoming Chen. Vortex documentation, 2025. URL https://infini-ai-lab.github.io/vortex_torch/.

Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, et al. Magicpig: Lsh sampling for efficient llm generation. *arXiv preprint arXiv:2410.16179*, 2024.

DeepSeek-AI. Deepseek-v3.2-exp: Boosting long-context efficiency with deepseek sparse attention, 2025.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures, 2018. URL https://arxiv.org/abs/1802.01561.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, et al. Areal: A large-scale asynchronous reinforcement learning system for language reasoning. *arXiv preprint arXiv:2505.24298*, 2025.

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

K Han, A Gu, WD Li, F Yan, T Zhang, S Wang, A Solar-Lezama, K Sen, and I Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Horace He and Thinking Machines Lab. Defeating nondeterminism in llm inference. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20250910. https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/.

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, 2025. URL https://arxiv.org/abs/2508.18588.

Jian Hu, Xibin Wu, Zilin Zhu, Xianyu, Weixun Wang, Dehao Zhang, and Yu Cao. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework. *arXiv preprint arXiv:2405.11143*, 2024.

Wei Huang, Yi Ge, Shuai Yang, Yicheng Xiao, Huizi Mao, Yujun Lin, Hanrong Ye, Sifei Liu, Ka Chun Cheung, Hongxu Yin, Yao Lu, Xiaojuan Qi, Song Han, and Yukang Chen. Qerl: Beyond efficiency – quantization-enhanced reinforcement learning for llms, 2025. URL https://arxiv.org/abs/2510.11696.

Infini-AI-Lab. Vortex: A flexible and efficient sparse attention framework, 2026. URL https://github.com/Infini-AI-Lab/vortex_torch. GitHub repository.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023. URL https://arxiv.org/abs/2211.17192.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. Taco: Topics in algorithmic code generation dataset. *arXiv preprint arXiv:2312.14852*, 2023a.

Ziniu Li, Tian Xu, Yushun Zhang, Zhihang Lin, Yang Yu, Ruoyu Sun, and Zhi-Quan Luo. Remax: A simple, effective, and efficient reinforcement learning method for aligning large language models. *arXiv preprint arXiv:2310.10505*, 2023b.

Bingshuai Liu, Ante Wang, Zijun Min, Liang Yao, Haibo Zhang, Yang Liu, Xu Han, Peng Li, Anxiang Zeng, and Jinsong Su. Spec-rl: Accelerating on-policy reinforcement learning with speculative rollouts, 2026. URL https://arxiv.org/abs/2509.23232.

Hongyi Liu, Zhuoming Chen, Yang Zhou, Haizhong Zheng, and Beidi Chen. Jackpot: Optimal budgeted rejection sampling for extreme actor-policy mismatch rl. https://github.com/Infini-AI-Lab/jackpot.git, 2025a. Official GitHub repository.

Jiawei Liu and Lingming Zhang. Code-r1: Reproducing r1 for code with reliable rewards. GitHub repository, 2025. https://github.com/ganler/code-r1.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=1qvx610Cu7.

Liyuan Liu, Feng Yao, Dinghuai Zhang, Chengyu Dong, Jingbo Shang, and Jianfeng Gao. Flashrl: 8bit rollouts, full power rl, August 2025b. URL https://fengyao.notion.site/flash-rl.

Xiaoxuan Liu, Jongseok Park, Langxiang Hu, Woosuk Kwon, Zhuohan Li, Chen Zhang, Kuntai Du, Xiangxi Mo, Kaichao You, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Turbospec: Closed-loop speculation control system for optimizing llm serving goodput, 2025c. URL https://arxiv.org/abs/2406.14066.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, Zhiqi Huang, Huan Yuan, Suting Xu, Xinran Xu, Guokun Lai, Yanru Chen, Huabin Zheng, Junjie Yan, Jianlin Su, Yuxin Wu, Neo Y. Zhang, Zhilin Yang, Xinyu Zhou, Mingxing Zhang, and Jiezhong Qiu. Moba: Mixture of block attention for long-context llms, 2025. URL https://arxiv.org/abs/2502.13189.

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. https://pretty-radio-b75.notion.site/DeepScaleR.

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. *arXiv preprint arXiv:2601.10079*, 2026.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, Benjamin Bossan, and Marian Tietz. PEFT: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft, 2022.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich, Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani, Bohan Zhang, Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao, Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi, Cary Bassin, Cary Hudson, Chak Ming Li, Charles de Bourcy, Chelsea Voss, Chen Shen, Chong Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts, Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely, David Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong, Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang, Felipe Petroski Such, Filippo Raso, Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace Zhao, Greg Brockman, Guillaume Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin, Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian Osband, Ignasi Clavera Gilaberte, Ilge Akkaya, Ilya Kostrikov, Ilya Sutskever, Irina Kofman, Jakub Pachocki, James Lennon, Jason Wei, Jean Harb, Jerry Twore, Jiacheng Feng, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joaquin Quiñonero Candela, Joe Palermo, Joel Parish, Johannes Heidecke, John Hallman, John Rizzo, Jonathan Gordon, Jonathan Uesato, Jonathan Ward, Joost Huizinga, Julie Wang, Kai Chen, Kai Xiao, Karan Singhal, Karina Nguyen, Karl Cobbe, Katy Shi, Kayla Wood, Kendra Rimbach, Keren Gu-Lemberg, Kevin Liu, Kevin Lu, Kevin Stone, Kevin Yu, Lama Ahmad, Lauren Yang, Leo Liu, Leon Maksin, Leyton Ho, Liam Fedus, Lilian Weng, Linden Li, Lindsay McCallum, Lindsey Held, Lorenz Kuhn, Lukas Kondraciuk, Lukasz Kaiser, Luke Metz, Madelaine Boyd, Maja Trebacz, Manas Joglekar, Mark Chen, Marko Tintor,

Mason Meyer, Matt Jones, Matt Kaufer, Max Schwarzer, Meghan Shah, Mehmet Yatbaz, Melody Y. Guan, Mengyuan Xu, Mengyuan Yan, Mia Glaese, Mianna Chen, Michael Lampe, Michael Malek, Michele Wang, Michelle Fradin, Mike McClay, Mikhail Pavlov, Miles Wang, Mingxuan Wang, Mira Murati, Mo Bavarian, Mostafa Rohaninejad, Nat McAleese, Neil Chowdhury, Neil Chowdhury, Nick Ryder, Nikolas Tezak, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Patrick Chao, Paul Ashbourne, Pavel Izmailov, Peter Zhokhov, Rachel Dias, Rahul Arora, Randall Lin, Rapha Gontijo Lopes, Raz Gaon, Reah Miyara, Reimar Leike, Renny Hwang, Rhythm Garg, Robin Brown, Roshan James, Rui Shu, Ryan Cheu, Ryan Greene, Saachi Jain, Sam Altman, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Santiago Hernandez, Sasha Baker, Scott McKinney, Scottie Yan, Shengjia Zhao, Shengli Hu, Shibani Santurkar, Shraman Ray Chaudhuri, Shuyuan Zhang, Siyuan Fu, Spencer Papay, Steph Lin, Suchir Balaji, Suvansh Sanjeev, Szymon Sidor, Tal Broda, Aidan Clark, Tao Wang, Taylor Gordon, Ted Sanders, Tejal Patwardhan, Thibault Sottiaux, Thomas Degry, Thomas Dimson, Tianhao Zheng, Timur Garipov, Tom Stasi, Trapit Bansal, Trevor Creech, Troy Peterson, Tyna Eloundou, Valerie Qi, Vineet Kosaraju, Vinnie Monaco, Vitchyr Pong, Vlad Fomenko, Weiyi Zheng, Wenda Zhou, Wenting Zhan, Wes McCabe, Wojciech Zaremba, Yann Dubois, Yinghai Lu, Yining Chen, Young Cha, Yu Bai, Yuchen He, Yuchen Zhang, Yunyun Wang, Zheng Shao, and Zhuohan Li. Openai o1 system card, 2026. URL https://arxiv.org/abs/2412.16720.

Alexandre Piché, Ehsan Kamalloo, Rafael Pardinas, Xiaoyin Chen, and Dzmitry Bahdanau. Pipelinerl: Faster on-policy reinforcement learning for long sequence generation, 2025. URL https://arxiv.org/abs/2509.19128.

Penghui Qi, Zichen Liu, Xiangxin Zhou, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Defeating the training-inference mismatch via fp16, 2025. URL https://arxiv.org/abs/2510.26788.

Ranajoy Sadhukhan, Zhuoming Chen, Haizhong Zheng, Yang Zhou, Emma Strubell, and Beidi Chen. Kinetics: Rethinking test-time scaling laws, 2025. URL https://arxiv.org/abs/2506.05333.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

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*, pp. 1279–1297, 2025a.

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*, pp. 1279–1297, 2025b.

Hongjin Su, Shizhe Diao, Ximing Lu, Mingjie Liu, Jiacheng Xu, Xin Dong, Yonggan Fu, Peter Belcak, Hanrong Ye, Hongxu Yin, et al. Toolorchestra: Elevating intelligence via efficient model and tool orchestration. *arXiv preprint arXiv:2511.21689*, 2025.

Qidong Su, Christina Giannoula, and Gennady Pekhimenko. The synergy of speculative decoding and batching in serving large language models, 2023. URL https://arxiv.org/abs/2310.18813.

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. *arXiv preprint arXiv:2410.21465*, 2024a.

Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding, 2024b. URL https://arxiv.org/abs/2404.11912.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024a.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference, 2024b. URL https://arxiv.org/abs/2406.10774.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. Trl: Transformer reinforcement learning. https://github.com/huggingface/trl, 2020.

Yongji Wu, Xueshen Liu, Haizhong Zheng, Juncheng Gu, Beidi Chen, Z Morley Mao, Arvind Krishnamurthy, and Ion Stoica. Rlboost: Harvesting preemptible resources for cost-efficient reinforcement learning on llms. *arXiv preprint arXiv:2510.19225*, 2025.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

Ran Yan, Youhe Jiang, and Binhang Yuan. Flash sparse attention: More efficient natively trainable sparse attention. *arXiv preprint arXiv:2508.18224*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention, 2025a. URL https://arxiv.org/abs/2502.11089.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23078–23097, 2025b.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2o: Heavy-hitter oracle for efficient generative inference of large language models, 2023. URL https://arxiv.org/abs/2306.14048.

Haizhong Zheng, Jiawei Zhao, and Bedi Chen. Prosperity before collapse: How far can off-policy rl reach with stale data on llms? *arXiv preprint arXiv:2510.01161*, 2025.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs, 2024. URL https://arxiv.org/abs/2312.07104.

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, 2025. URL https://arxiv.org/abs/2509.18521.

APPENDIX CONTENTS

This appendix provides additional analyses, implementation details, and supporting empirical evidence for the main paper. The sections are organized as follows:

Appendix A: Sparse Rollout Instability and High-Reward Rollouts (p. 16)
We examine whether insufficient rollout quality is the primary cause of sparse rollout RL instability, and show that this factor alone does not explain training collapse.

Appendix B: Extended Related Work (p. 17)
We provide a more detailed discussion of prior work on RLVR, sparse attention, efficient long-context inference, and policy mismatch in reinforcement learning.

Appendix C: Cost Model Implementation and Visualization (p. 18)
We describe the implementation details of our rollout cost model and provide additional visualizations illustrating the efficiency tradeoffs of sparse attention.

Appendix D: Empirical Overhead of DISTILLSPARSE LoRA Distillation (p. 19)
We quantify the additional training overhead introduced by LoRA-based distillation and analyze its practical impact on end-to-end efficiency.

Appendix E: Distribution Mismatch under Sparse Rollout Collapse (p. 20)
We present additional evidence characterizing the distributional mismatch between sparse and dense rollout policies during unstable RL training.

Appendix F: Rollout Cost Analysis and Sparse Attention Motivation (p. 21)
We further motivate sparse attention rollouts through a detailed analysis of rollout-stage computational cost.

Appendix G: Acceptance-Rate Definition and Top-k Approximation (p. 22)
We define overlap and mismatch precisely and evaluate the truncated-probability approximation.

Appendix H: Tail-Quantile Sensitivity and Training Variability (p. 23)
We compare three tail quantiles and report three independent coding-training runs.

Appendix I: Comparison and Composition with Other Rollout Optimizations (p. 23)
We compare FP8 KV-cache quantization and dynamic sparsity, and evaluate asynchronous RL.

Appendix J: Transfer across Model Scale, Task Domain, and Model Family (p. 24)
We report full-epoch 14B, coding-domain, and Llama transfer experiments.

Appendix K: Calibration, KV-Budget Trends, and Applicability (p. 25)
We describe profiling cost, measured budget trends, and limits of architectural transfer.

Appendix L: Full-Run DistillSparse and Integration Limits (p. 26)
We report the two-stage full run, LoRA configuration, and remaining scaling and asynchronous-integration questions.

Appendix M: Lower-Tail Distribution and Beta-Fit Diagnostics (p. 28)
We provide the acceptance-rate tail visualization, Beta fits, and Kolmogorov–Smirnov statistics.

Appendix N: Online LoRA Distillation Algorithm (p. 29)
We provide the complete DistillSparse update procedure, including separate dense-policy and sparse-adapter updates.

LIMITATIONS

Our study focuses on sparse rollout for RL with verifiable rewards on a limited set of base models, tasks, and hardware settings, so the exact stability thresholds and speedup tradeoffs we report may not transfer unchanged to other model families, reward formulations, or deployment environments. In particular, our conclusions are tied to long-context reasoning workloads and to the sparse-attention configurations we evaluate, and broader validation across additional domains such as multimodal RL, alternative policy optimization algorithms, and more diverse system stacks would be needed before treating the proposed acceptance-rate thresholds and scheduling rules as universal design principles.

BROADER SOCIAL IMPACT

If successful, our method can reduce the compute cost of RL training for large language models, which may lower the barrier to studying reasoning systems and make experimentation more accessible to smaller research groups; at the same time, cheaper post-training can also accelerate the development and deployment of increasingly capable models, including systems that may be misused for generating persuasive misinformation, scalable spam, or unsafe automated assistance. We therefore view this work primarily as a systems contribution whose social impact depends on how downstream models are evaluated and released, and we encourage practitioners to pair efficiency gains with careful capability assessment, application-specific safeguards, and responsible release practices.

A INSUFFICIENT ROLLOUT IS NOT THE MAIN CULPRIT OF SPARSE ROLLOUT RL INSTABILITY

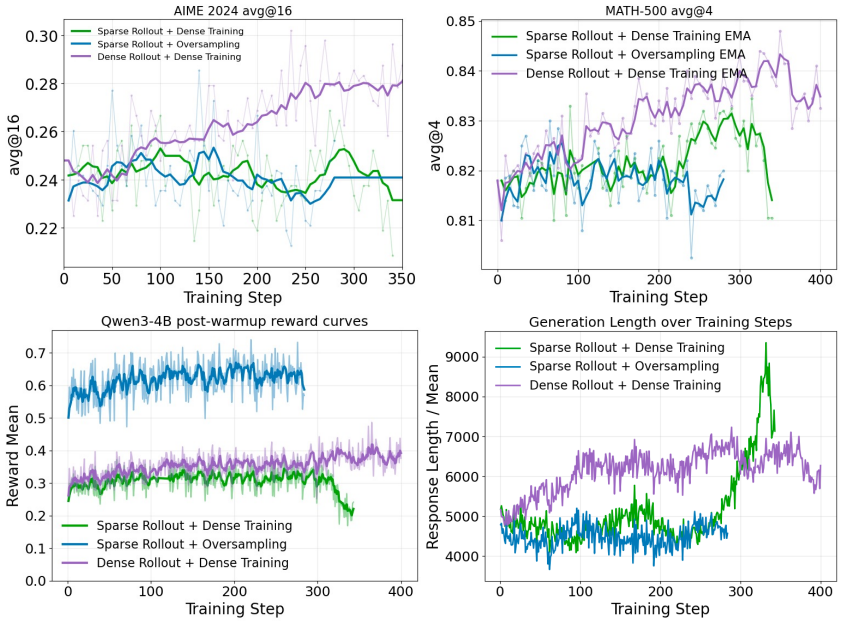

Figure 7: We illustrate with a simple experiment that sparse rollout RL training dense policy collapse is not only due to the reward is insufficient. Oversampling (Sample 16 select Top-8 based on zero-one-reward) drastically increases reward of the rollout but still cannot prevent the training performance to be near the dense rollout.

In this section, we present a concrete study showcasing that although sparse rollout decreases the average RL training reward, the insufficient reward alone does not cause the RL training to crash.

To save computational resources, we don't use the thinking models as we did in the main experiment section, as training on 37K generation length makes rollout extremely cost-heavy. Instead, we base our experiments on top of base models. To make the model CoT generation length longer. We first warmup the standard base models on DeepScaleR dataset Luo et al. (2025) for 20K training examples so that the model average generation length raises to above 6K tokens, from where we then train the model on POLARIS dataset An et al. (2025).

Specifically, for Qwen3-4B-Base model we underwent the above warmup and then start the training with sparse rollout with 12K generation length cutoff. We present RL training trials in Figure 7. We use `pagesize` 16 and `number of pages` 32 for sparse rollout. We compare sparse-rollout RL training (green) with the dense-rollout baseline (purple). On AIME2024 and MATH500, the green curve is consistently below the purple curve. Also note that the sparse-rollout training reward is initially only marginally lower than dense, but eventually crashes.

For sparse rollout, we attempted to increase its average training rewards. In fact, for every training example problem we sample 16 times (instead of N = eight times in green curve), then we rank the 16 rollout trajectories per example problem by their training reward and select the top-8. The average reward curve is shown in blue, which is significantly higher than dense average reward and that of the original sparse rollout. However, the blue curve still fails to recover the training performance.

B Extended Related Works

In this section, we delve in the most relevant related works to our study and cleanly present the problem of interest. Reinforcement learning methods for LLMs include ReMax (Li et al., 2023b), RLOO (Ahmadian et al., 2024), and GRPO (Shao et al., 2024); systems such as Verl (Sheng et al., 2025b), AReal (Fu et al., 2025), TRL (von Werra et al., 2020), and OpenRLHF (Hu et al., 2024) support scalable training.

- **Prior Rollout Speedup Methods.** Despite many recent works proposed to reduce the cost of LLM RL, they are with key limitations. (1) **asynchronous RL** (Zheng et al., 2025; Piché et al., 2025; Zhou et al., 2025) training systems are proposed accelerate training by decoupling the rollout and training phases to remove bubbles and achieve higher compute utilization. However, they cannot reduce the underlying long-context generation workload; when the rollout workload is large, training can remain slow even at high utilization. (2) while **model quantization in RL** (Liu et al., 2025b; Huang et al., 2025)are proposed to reduce the cost of loading model weights, it cannot effectively mitigate the rollout overhead for long-sequence generation, where KV-cache loading remains the primary bottleneck (Sadhukhan et al., 2025). (3) Conversely, **speculative decoding** (SpecDec) (Leviathan et al., 2023; Chen et al., 2023; Liu et al., 2026; He et al., 2025) can accelerate rollouts without altering the sampling distribution. Still, they suffer from weaknesses. First, the SpecDec techniques requiring draft models cannot achieve substantial practical speedup under large batchsize setting, as verification becomes compute-intensive per (Liu et al., 2025c; Su et al., 2023). Second, the SpecDec techniques using past epochs rollout generation as draft to speedup new epoch rollout are not always practically applicable, as the realistic industry RL training are usually done in one epoch.

- **Sparse attention** is a promising solution to speedup RL rollout. First, during RL training in practice of the thinking models, long-context rollout accounts for over 70% of per-step cost (Wu et al., 2025). Second, sparse attention has been extensively studied and proved to be effective in achieving significant speedup with great preservation of generation quality. (Xiao et al., 2024; Zhang et al., 2023; DeepSeek-AI, 2025; Chen et al., 2024) For our study, we zoom in on one particular type of sparse attention family, **block-sparse attention** as a case study (Lu et al., 2025; Yuan et al., 2025a; Tang et al., 2024a; Sun et al., 2024b).[5] Abitrary configuring sparsity in rollout is highly prone to training collapse, and as shown in Figure 2(a) the response length tends to tank during collapse.[6] However, prior works in sparse attention over-emphasize on generation accuracy, which is very different from the true cause of RL collapse, per-token actor-policy distribution mismatch. We showcase the above by runing the following experiment. In Appendix A we show that under sparse rollout that triggers dense policy collapse, even if we apply test-time-scaling and filtering to drastically increase the average rewards, even above those of dense rollout, the training performance is still not recovered.

- **Algorithmic Techniques on Actor-Policy Alignment** have also been extensively studied, but they cannot provide general solution for maintaining sparse rollout dense training RL stability. These techniques are either Importance-sampling based (Espeholt et al., 2018; Fu et al., 2025; Liu et al., 2025b) or Rejection-sampling based (Liu et al., 2025a; Qi et al., 2025; Luo et al., 2026). Complementary systems techniques, including FP32 LM heads (Liu et al., 2025b) and deterministic inference (He & Lab, 2025), mitigate numerical discrepancies during rollout. However, these techniques prioritizes conventional actor-policy mismatch scenarios staleness. However, we show in Appendix F that sparse rollout introduces failure mode which induce orders of magnitude larger KL divergence between actor and policy larger than staleness problems, where these techniques succeed.

[5] We believe similar conclusion about sparse rollout dense training can be drawn in fine-grained sparse-attention works.

[6] For the experiment, we train Qwen3-1.7B thinking model with 37K generation length on Polaris with KV budget 1024.

Thus, these methods fail to stablize the training shown in Figure 2(a).[7] Admittedly, we can drastically increase truncation/rejection threshold for more stability, but will hurting convergence. **Therefore, to realistically unlock the efficiency benefits of sparse attention in long-context rollout, we need to systematically study the lowest sparse-rollout cost that still enables stable dense-model training, as outlined in the next sections.**

C Detailed Implementation and Visualization of the Cost Model

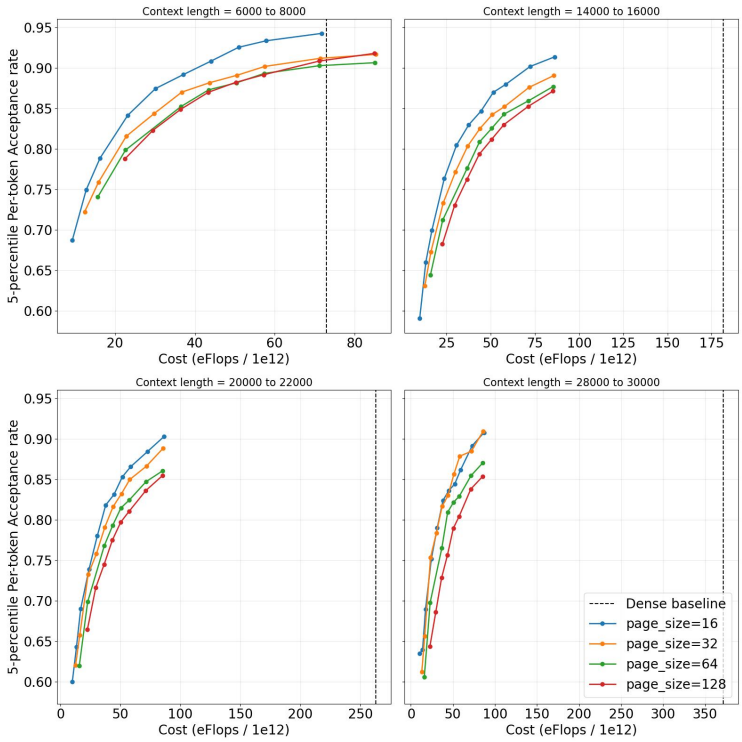

Figure 8: Additional cost analysis showing how different page sizes compare across generation-length regimes.

In this section, we provide more details on the cost-model implementation and visualize how `pagesize` relates to cost and distribution mismatch with the dense model.

Specifically, for the cost model analysis, we use Qwen3-1.7B model configurations. In particular, we look at `parallel_sampling` to be 8, mimicking the conventional GRPO rollout scenario, and base the analysis on one H200, similar takeaways can also be drawn on B200 as well. (Specifically, we use arithmetic intensity based on H200 but for our conclusion the arithmetic intensity holds when we use B200 arithmetic intensity.) For each `pagesize` setting, we pick 11 KV-budget configurations, covering tail acceptance rates of 0.7 to 0.9 across all experimental sequence lengths.

Table 4: Empirically, we also observe a similar pattern: the page size from 16 and above does not affect the decoding speed a lot as long as the sparse engine's Top-k kernel is well implemented (Cost measured on single H200 with batch size 16 in ms).

Vortex configs (`pagesize-num_page`)	Past KV Size 16k	24k	32k
16-64	4.59	5.13	5.41
32-32	4.47	4.96	5.30
64-16	4.30	4.87	5.12
128-8	4.13	4.84	5.07

[7] We use TIS to correct Qwen3-1.7B thinking RL with sparse rollout with KV budget of 1024 and 37K generation length cutoff. In Appendix E, we also show that rejection-sampling based method also cannot solve the instability under the same condition.

There are two aspects where we think are worth highlighting. First, `pagesize`=16 occupies the Pareto frontier across all generation lengths. However, as generation length increases, the gap between `pagesize`=32 and 16 shrinks; by 30K tokens, `pagesize`=32 nearly reaches the Pareto frontier and is almost on par with `pagesize`=16 in the cost–tail-acceptance-rate tradeoff. Second, as the generation sequence length increases, the gap between cost of using sparse attention rollout and dense attention increases, showing more cost benefits from using sparse attention.

Benefitted from the high-quality top-k kernels supported in Infini-AI-Lab (2026), we are able to empirically show that scoring portion of the sparse attention cost in the model generation is not significantly impacted by the page size selection. Shown in Table 4, we measure the average generation cost in the full forward pass of the sparse attention model under different `pagesize` selections. Particularly, we use a single H200 GPU, and we prefill with sequence lengths 16K, 24K, 32K sequences with batch size 8, and compute the average full-model decoding pass by averaging over 1000 trials. The average is presented in milliseconds (ms).

D Empirical Study of DistillSparse Overhead of LoRA Distillation Training

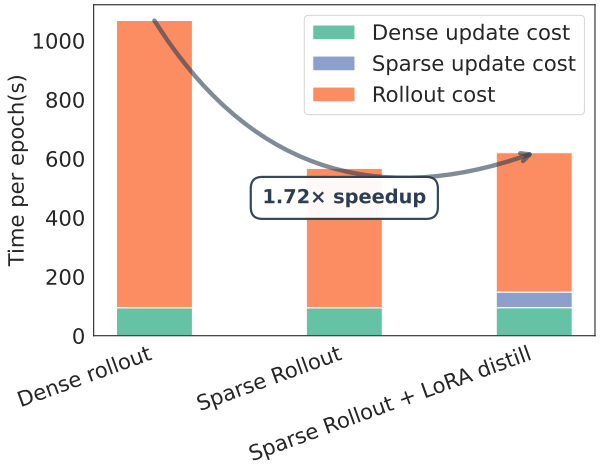

Figure 9: Per-epoch time breakdown for Qwen3-4B-Instruct (16K generation length, 2×H200) under dense rollout, sparse rollout, and sparse rollout with LoRA distillation (DistillSparse). DistillSparse achieves a 1.72× end-to-end speedup over dense rollout.

System implementation & Efficiency: In this section, we show that DistillSparse incurs minimal overhead. To simplify the implementation, for the speedup demonstration, we choose a simplified setting, where instead of using a sparsity schedule for rollout and training with dense distillation, we instead use a fixed sparse budget.

To conduct our experiments, we use Verl (Sheng et al., 2025a) with FSDP as the training engine and SGLang (Zheng et al., 2024) as the inference engine. Experiments are run on Qwen3-4B-Instruct with generation length 16K. Training is run on 2xH200 GPUs. For efficient sparse-attention rollouts, we use Vortex_torch (Chen, 2025). We adopt block top-k attention with a page size of 16, and set the number of top-k pages according to the sparse KV budget. In addition, we use Flash Sparse Attention (Yan et al., 2025) for efficient sparse-attention training and PEFT (Mangrulkar et al., 2022) for LoRA adaptation.

As shown in Figure 9, we report the efficiency of our implementation. When training 4B instruct model with 16K max context length, dense rollouts account for roughly 90% of the per-epoch time. Sparse attention directly alleviates this bottleneck and accelerates rollouts by roughly 2.1×, while the added sparse distillation only modestly increases the policy-update cost. Overall, DistillSparse achieves a 1.72× **end-to-end speedup**, showing that its overhead is small.

E DISTRIBUTION MISMATCH UNDER SPARSE ROLLOUT TRAINING COLLAPSE: VISUALIZATION OF KL DIVERGENCE OF ACTOR-POLICY MISMATCH

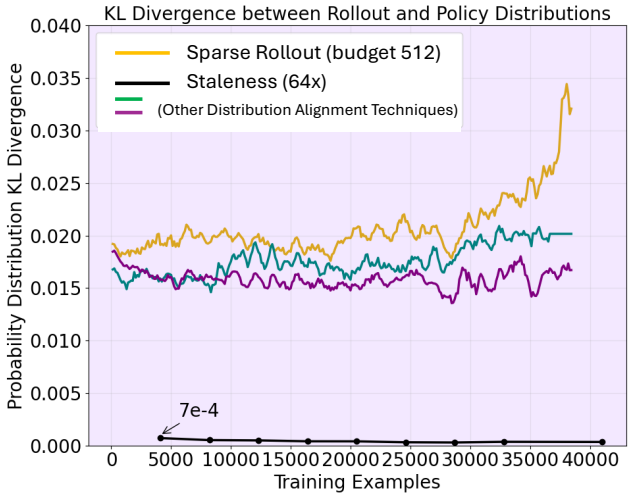

Figure 10: Comparison of actor-policy KL Divergence between actor-policy for Qwen3-4B-Base models under 64 times staleness and sparse to dense training.

In this section, we show an illustration of why sparse-dense RL training causes such a huge problem when selecting the sparsity in ad hoc manner. Compared to traditional scenarios such as aggressive staleness where the distribution alignment techniques like TIS (Liu et al., 2025b) and (Liu et al., 2025a), when choosing aggressive sparsity for rollout, the actor-policy divergence is much more severe, in particular the KL divergence is one order of magnitude higher.

Specifically, we measure the above visualization under the following settings. To save compute, we follow Section A to use Qwen3-4B-Base after warmup with sparse rollout (KV budget 512). With TIS + Jackpot applied, the resulting KL divergence is plotted in the Figure 10 yellow curve, in which it crashes in the later stage of training. In contrast, we also plot the KL divergence of the 64x staleness on Qwen3-4B-Base in black.

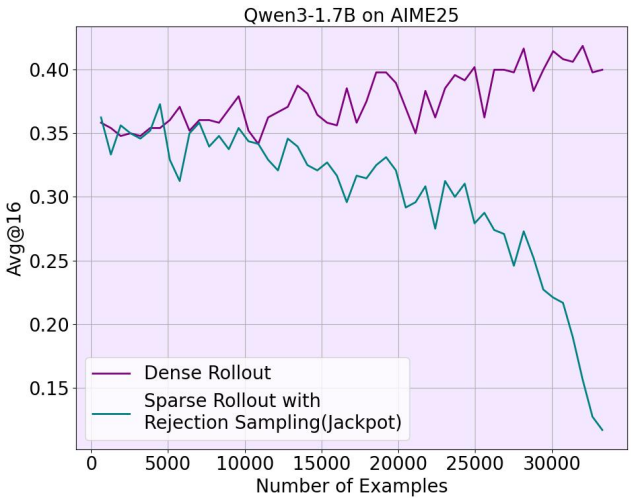

Figure 11: Besides Important-sampling alignment method, we show that rejection sampling (Jackpot) is also unable to solve training instability of 37K long-context rollout by itself.

The huge KL divergence renders the conventional alignment techniques unable to offer a general solution for sparse rollout training, motivating for a comprehensive study of the optimal tail acceptance-rate threshold and optimal sparsity configuration selection. For Qwen3-1.7B thinking under sparse rollout with a KV budget of 1024, TIS alone cannot solve the issue of training instability. Here Figure 11, we also provide the study of using the rejection-sampling method Liu et al. (2025a) on the same setting, and show that it also cannot repair instability.

F Rollout Cost Analysis and Sparse Attention Motivation

To understand why rollout generation dominates RL training, prior work models decoding cost as the combination of computation and KV-cache memory traffic (Sadhukhan et al., 2025). We adopt the same additive view and analyze rollout latency independently.

For dense autoregressive decoding with N sampled responses, we use the same cost model as Section 2.3, in eFLOPs, where I is the arithmetic intensity of the hardware. The compute cost consists of parametric computation plus self-attention:

$$C_{comp} = 2 \cdot P \cdot N \cdot L_{out} + r \cdot (2 \cdot L_{in} + L_{out}) \cdot L_{out} \cdot N \cdot D,$$

while memory traffic is driven by KV-cache access:

$$C_{mem} = 2 \cdot L_{in} \cdot L_{out} \cdot D + N \cdot L_{out}^2 \cdot D, \qquad C_{dense} = C_{comp} + I \cdot C_{mem}.$$

Although parameter access can be amortized with large batch size and model parallel inference, the attention term grows quadratically with response length. As a result, rollout decoding becomes increasingly memory-bound for long generations, and self-attention quickly emerges as the dominant bottleneck.

Sparse attention. With a KV budget B and block size `pagesize`, sparse decoding replaces the quadratic KV access with a budgeted cost plus a block-scoring cost:

$$C_{\text{sparse, no scoring}} = \underbrace{2 \cdot N \cdot P \cdot L_{out} + 2 \cdot r \cdot N \cdot D \cdot B \cdot L_{out}}_{\text{compute}} + \underbrace{2 \cdot I \cdot N \cdot D \cdot B \cdot L_{out}}_{\text{memory}},$$

$$C_{scoring} = \underbrace{2 \cdot N \cdot L_{in} \cdot D \cdot L_{out} + \frac{r \cdot N \cdot D \cdot L_{out}^2}{2 \cdot \text{pagesize}}}_{\text{compute}} + \underbrace{2 \cdot I \cdot L_{in} \cdot D \cdot L_{out} + \frac{I \cdot N \cdot D \cdot L_{out}^2}{2 \cdot \text{pagesize}}}_{\text{memory}},$$

$$C_{sparse} = C_{\text{sparse, no scoring}} + C_{scoring}.$$

Thus sparse attention replaces the $\mathcal{O}(L_{out}^2)$ KV access with $\mathcal{O}(BL_{out})$ growth, while the remaining quadratic scoring term is reduced by a factor of $2 \cdot$ `pagesize`.

End-to-End Implications: An RL iteration decomposes into rollout generation, logits recomputation, and policy update:

$$C_{\text{total}} = C^{\text{rollout}} + C^{\text{recomp}} + C^{\text{update}}.$$

Recomputation and update operate on full sequences via prefill-style passes, achieving high arithmetic intensity and remaining largely compute-bound. In contrast, rollout decoding performs per-token KV access with low arithmetic intensity, making it fundamentally memory-bound.

Superior Inference Scaling with Sparse Attention. Overall, the rollout phase dominates end-to-end training time (often $> 90\%$, Figure 9), primarily because the KV loading cost increases quadratically with generation length (Sadhukhan et al., 2025). Increasing number of rollouts cannot amortize the decoding cost as the dominant KV memory also grows linearly with it. Sparse attention directly targets this bottleneck by reducing rollout memory cost from $\mathcal{O}(L_{\text{out}}^2)$ to $\mathcal{O}(BL_{out})$, where B is the sparse KV budget. This allows us to increase the *generation length* and *number of samples* to achieve high-quality rollouts within reasonable cost. In other words, **sparse attention unlocks superior inference scaling in the usual long CoT regime.**

G ACCEPTANCE-RATE DEFINITION AND TOP-k APPROXIMATION

For a fixed generated prefix, let P and Q denote the dense-policy and rollout-policy next-token distributions over vocabulary $\mathcal{V}$. We define the acceptance rate as their probability overlap:

$$\alpha(P, Q) = \sum_{v \in \mathcal{V}} \min\{P(v), Q(v)\} = 1 - \mathrm{TV}(P, Q) = 1 - \tfrac{1}{2}\|P - Q\|_1. \tag{4}$$

Thus, the probability mismatch is $1 - \alpha$, not α itself. A 5th-percentile acceptance-rate target of 0.86 refers to a lower-tail statistic of overlap; it is not an L_1 distance of 0.86. Throughout these additional experiments, $\mathrm{DSparse}\tau$ denotes a dynamic sparsity schedule calibrated to a 5th-percentile acceptance-rate target τ, rather than a retained-KV fraction. Combining a schedule with another approximation can change its measured acceptance rate.

G.1 TRUNCATED OVERLAP AND OMITTED MASS

Let S_P and S_Q be the returned top-k token sets. We retain the returned probabilities without renormalizing, assigning zero to tokens absent from each respective set:

$$\widetilde{P}(v) = P(v)\mathbf{1}\{v \in S_P\}, \qquad \widetilde{Q}(v) = Q(v)\mathbf{1}\{v \in S_Q\}.$$

The truncated overlap is

$$\widehat{\alpha}_k = \sum_{v \in S_P \cup S_Q} \min\{\widetilde{P}(v), \widetilde{Q}(v)\}.$$

A token returned by only one distribution contributes zero; omitted mass is not redistributed. If $\epsilon_P = 1 - \sum_{v \in S_P} P(v)$ and $\epsilon_Q = 1 - \sum_{v \in S_Q} Q(v)$, then

$$0 \le \alpha - \widehat{\alpha}_k \le \min\{1, \epsilon_P + \epsilon_Q\}.$$

This follows by bounding the omitted overlap outside $S_P \cap S_Q$ by the omitted mass in the two distributions. High aggregate coverage alone does not guarantee uniformly small errors for every low-acceptance token, so we also measure sensitivity of the tail statistic to k.

Table 5: Top-k probability-mass coverage for Qwen3 thinking models. Coverage values are percentages.

Model	Tokens measured	Top-20	Top-40	Top-100
Qwen3-1.7B	31,967	99.995%	99.999%	99.999%
Qwen3-4B	32,768	99.992%	99.998%	99.999%
Qwen3-8B	32,768	99.997%	99.999%	99.999%

Table 6: Sensitivity of the overall 5th-percentile acceptance rate to the number of returned token probabilities. Valid-token counts differ between measurements.

k	Valid tokens	5th-percentile acceptance
20	775,191	94.8120%
40	793,432	94.8372%
100	759,549	94.8365%
1000	763,948	94.8214%

The measured tail statistics span only 0.0252 percentage points across these settings, supporting the use of top-20 in the evaluated Qwen3 setting. These measurements are a sensitivity check rather than a paired, token-by-token error analysis, because the valid-token counts differ. They also are not measurements at the 0.86 stability boundary. Top-20 is not assumed sufficient for every model family: the Llama-3.2-1B-Instruct experiment uses top-500 for adequate mass coverage.

H TAIL-QUANTILE SENSITIVITY AND TRAINING VARIABILITY

H.1 CORRESPONDENCE BETWEEN LOWER-TAIL QUANTILES

We measure the 5th, 20th, and 30th percentiles for schedules selected using 5th-percentile targets of 0.86 and 0.92. Table 7 reports their values in successive generation-length intervals. For DSparse0.86, the 20th percentile ranges from 0.94 to 0.96 and the 30th from 0.97 to 0.98. For DSparse0.92, these ranges are 0.97–0.98 and 0.99, respectively. Thus, the evaluated quantiles show a consistent correspondence and preserve the ordering between these schedules; their numerical thresholds are not interchangeable.

This comparison does not establish that the 5th percentile is more predictive than every alternative, nor does it report independent RL training sweeps selected using each quantile. We use the 5th percentile as a sensitive summary of the lower tail, while avoiding the most extreme quantiles whose estimates can have high variance.

Table 7: Acceptance-rate quantiles across generation length for two calibrated schedules. Each row gives an interval in generated tokens; all acceptance rates are fractions.

Length	0.86: 5th	20th	30th	0.92: 5th	20th	30th
0–2K	0.86	0.96	0.98	0.93	0.98	0.99
2–4K	0.86	0.95	0.98	0.93	0.97	0.99
4–6K	0.86	0.95	0.98	0.93	0.97	0.99
6–8K	0.84	0.94	0.97	0.92	0.97	0.99
8–10K	0.85	0.94	0.98	0.91	0.97	0.99
10–12K	0.85	0.95	0.98	0.92	0.97	0.99
12–14K	0.86	0.95	0.98	0.92	0.97	0.99
14–16K	0.86	0.96	0.98	0.91	0.97	0.99
16–18K	0.86	0.96	0.98	0.91	0.97	0.99
18–20K	0.86	0.96	0.98	0.92	0.98	0.99

H.2 THREE-SEED CODING EXPERIMENT

We repeat Qwen3-1.7B training on TACO with three random seeds, using DSparse0.86, a 24K maximum generation length, and one third of an epoch per run. Table 8 reports exponential-moving-average (EMA) training statistics and LiveCodeBench scores. LiveCodeBench ranges from 0.375 to 0.386, with a mean of approximately 0.380. The runs show similar behavior in this configuration. They do not quantify seed variability of the full threshold boundary, or establish seed robustness for every model and domain.

Table 8: Three random seeds for Qwen3-1.7B coding RL. Response length is measured in tokens; KL is relative to the old policy.

Run	EMA reward	EMA length	EMA entropy	EMA KL to old	LiveCodeBench
1	0.3539	10,961	0.3050	8.5×10^{-5}	0.378
2	0.3460	11,107	0.3029	8.5×10^{-5}	0.386
3	0.3409	10,681	0.3024	8.1×10^{-5}	0.375

I COMPARISON AND COMPOSITION WITH OTHER ROLLOUT OPTIMIZATIONS

I.1 FP8 KV-CACHE QUANTIZATION

We compare dynamic sparsity with FP8 KV-cache quantization, and use the acceptance-rate criterion to select combined configurations. On Qwen3-1.7B, we train on 20K Polaris examples with a 37K maximum generation length. FP8 KV-cache quantization alone has a trajectory-average 5th-percentile acceptance rate of 0.865. Combining FP8 with DSparse0.93 yields 0.860; the combined value differs

Table 9: Qwen3-1.7B comparison on Polaris. AIME scores use avg@16 and AMC uses avg@4. Throughput is reported in tokens/s on H200; speedups are relative to this experiment's dense baseline.

Setting	EMA reward	AIME24	AIME26	AMC	tok/s	Speedup
Dense	0.5424	0.5187	0.4354	0.7891	2417	1.00×
KV FP8	0.5419	0.5333	0.4312	0.7951	3000	1.24×
KV FP8 + DSparse0.93	0.5343	0.5166	0.4187	0.7981	3250	1.34×
DSparse0.86	0.5416	0.5166	0.4270	0.8012	5294	2.19×

from the sparsity schedule's standalone target. Both configurations remain stable in training. Table 9 shows the reward, downstream evaluations, and rollout throughput.

Dynamic sparsity alone provides the highest throughput among the evaluated 1.7B configurations. Combining optimizations is therefore not automatically advantageous. For Qwen3-4B and Qwen3-8B, Table 10 reports a further throughput comparison. At 4B, DSparse0.86 remains fastest. At 8B, FP8 with DSparse0.88 reaches a measured acceptance statistic of 0.86 and $2.54\times$ speedup, exceeding either technique alone. This table reports profiling results; it does not supply downstream training evaluations for these combined 4B and 8B configurations.

Table 10: Composition of FP8 and dynamic sparsity at 4B and 8B. Acceptance denotes the trajectory-average 5th-percentile statistic. Speedups are relative to the dense baseline of the same model.

Model	Setting	Acceptance	tok/s	Speedup
4B	Dense	—	1369	1.00×
4B	KV FP8	0.90	2443	1.78×
4B	KV FP8 + DSparse0.88	0.86	2922	2.13×
4B	DSparse0.86	0.86	3308	2.41×
8B	Dense	—	1300	1.00×
8B	KV FP8	0.90	2427	1.86×
8B	KV FP8 + DSparse0.88	0.86	3309	2.54×
8B	DSparse0.86	0.86	2682	2.06×

I.2 Compatibility with asynchronous RL

Following an AReaL-style setting with maximum staleness 16, we train Qwen3-1.7B for 10K examples with a 20K maximum generation length. Adding DSparse0.91 increases rollout throughput from 3421 to 5439 tokens/s, approximately $1.59\times$. Training reward and downstream scores are given in Table 11. This experiment evaluates dynamic sparsity with asynchronous RL; it does not evaluate DistillSparse or delayed LoRA updates.

Table 11: Asynchronous Qwen3-1.7B training with and without dynamic sparsity. AIME uses avg@16 and AMC uses avg@4.

Setting	EMA reward	AIME24	AIME26	AMC	tok/s
Async	0.5254	0.4708	0.3875	0.7771	3421
Async + DSparse0.91	0.5087	0.5000	0.3895	0.7620	5439

J Transfer across Model Scale, Task Domain, and Model Family

J.1 Full-epoch Qwen3-14B training

We extend Qwen3-14B Thinking training to more than 50K examples, using four H200 nodes (32 H200 GPUs) and approximately 190 hours. A dynamic sparsity schedule calibrated to the 0.86 target maintains stable training and provides approximately $1.5\times$ rollout speedup. Table 12 reports downstream evaluation. The sparse and dense results are close across these benchmarks, with differences in both directions.

Table 12: Full-epoch Qwen3-14B Thinking evaluation. AIME columns use Mean@16; AMC columns use Mean@4.

Rollout	AIME25	AIME26	AIME24	AMC22/23	AMC12
Dense	0.7500	0.7770	0.8125	0.8055	0.9396
DSparse0.86	0.7500	0.7708	0.8028	0.8111	0.9367

J.2 Transfer from mathematics to coding

We train on TACO with Qwen3-1.7B and Qwen3-4B using maximum generation lengths of 24K and 32K, respectively. The evaluation covers 20K training examples for 1.7B and 10K for 4B. We reuse the schedules calibrated on mathematics rather than searching for coding-specific schedules. Table 13 shows similar downstream performance to dense rollout. These results support schedule reuse between the evaluated long-context reasoning tasks, rather than universal transfer to arbitrary domains.

Table 13: Coding-domain transfer using mathematics-calibrated schedules.

Model	Rollout	Examples	LiveCodeBench	HumanEval+	MBPP+
1.7B	Dense	20K	0.3970	0.6219	0.5846
1.7B	DSparse0.86	20K	0.3933	0.6280	0.6084
4B	Dense	10K	0.5542	0.7228	0.7271
4B	DSparse0.86	10K	0.5551	0.7268	0.7301

J.3 Transfer to Llama

For Llama-3.2-1B-Instruct, we search for a model-specific sparsity schedule using the same 0.86 acceptance-rate target. We train for 22K examples on the MATH and GSM8K training sets with a 10K maximum generation length, using top-500 probabilities for acceptance-rate estimation. Training remains stable, and Table 14 reports the resulting benchmark scores. This experiment transfers the target, not the Qwen3 KV-budget schedule.

Table 14: Llama-3.2-1B-Instruct evaluation after 22K training examples.

Rollout	GSM8K	MATH500	AMC
Dense	0.519	0.304	0.1445
DSparse0.86	0.547	0.310	0.1686

The 0.86 criterion remains empirical. Truncated importance sampling (TIS), used throughout these training settings, offers a possible explanation: sufficiently high lower-tail alignment may keep mismatch within a range that TIS can correct effectively. This is an interpretation of the evidence, not a theoretical guarantee. These experiments do not establish transfer to tool use, agentic workflows, arbitrary RL objectives, or all model families.

K Calibration Procedure, KV-Budget Trends, and Applicability

K.1 Practical calibration and its cost

In a new setting using TIS with moderate clipping thresholds in the range 3–5, we use 0.86 as an initial acceptance-rate target. Practitioners should profile on representative prompts, inspect the 5th-percentile statistic across generation-length bins as well as its trajectory average, and adjust KV budgets or other approximation parameters to meet the target. The target can transfer even when the required sparsity schedule does not. A short validation run remains useful before committing to a long training run.

The calibration experiment generates sparse rollouts from 720 randomly sampled AIME problems and prefills these trajectories through the dense model to estimate acceptance rates. The full procedure

takes approximately 30 minutes. In the evaluated setting, estimates stabilize after 64 trajectories, reducing calibration to approximately 10 minutes on one H200 node. These measurements describe the evaluated configuration; they are not a scaling guarantee for larger models or longer contexts. The mathematics-to-coding transfer in Appendix J suggests that some related tasks can reuse calibration.

K.2 Measured KV-budget trends

We fit a monotonic saturating logistic curve for Qwen3-1.7B and Hill curves for Qwen3-4B and Qwen3-8B. Table 15 summarizes the fitted trends. The 1.7B budget reaches approximately 256 by 4K context and remains unchanged through 8K. For 4B and 8B, the additional budget required per 1K tokens decreases between 4K and 8K context. These observations indicate diminishing marginal budget growth within the measured range, rather than proportional growth with context length.

Table 15: KV-budget trends required to maintain the 0.86 target.

Model	Quantity	At 4K context	At 8K context
1.7B	Required budget	≈ 256	≈ 256
4B	Additional budget per 1K tokens	61.7	45.2
8B	Additional budget per 1K tokens	56.6	25.3

We do not infer a model-size scaling law or extrapolate the saturating fits beyond the measured range.

Even with sublinear KV-budget growth in a measured context range, rollout speedups can decrease with model size: parameter computation, MLPs, communication, synchronization, and other engine costs can occupy a larger share of generation time. Reducing attention cost does not eliminate those contributions.

K.3 Other attention architectures

The efficiency experiments do not include head-to-head evaluations against speculative decoding or native sparse-attention architectures. The acceptance-rate statistic compares rollout and training distributions and is not inherently restricted to one sparse-attention operator. It may help diagnose more aggressive rollout configurations for native sparse, hybrid-attention, or KV-sharing models, but these extensions require architecture-specific calibration and are not evaluated here. The present KV-selection scheduler does not directly apply to linear-attention models without explicit KV selection. Distributional overlap may still be informative when rollout and training use different approximations, but the 0.86 target is not established for those architectures.

L Full-Run DistillSparse and Integration Limits

L.1 Configuration and two-stage training

We evaluate a full DistillSparse run on Qwen3-1.7B Thinking with 46K training examples (two stages of 23K training examples each) and a 37K maximum generation length. The LoRA configuration uses rank 32, scaling parameter 64, and adapters on all linear layers. The run uses two stages: DSparse0.86 followed by DSparse0.80.

The trajectory-level 5th-percentile acceptance rate measured for the more aggressive DSparse0.80 schedule increases from 0.8182 at the start of stage 1 to 0.8609 after the first 23K training examples. After reaching this alignment, training switches directly to DSparse0.80. The final stage provides approximately $2.5\times$ rollout speedup relative to dense generation, compared with $2.2\times$ in the initial DSparse0.86 stage. These are stage-specific rollout speedups, not the end-to-end speedup of the entire run.

Table 16 provides the full downstream comparison. DistillSparse is higher on AIME25 and equal on the AMC metrics, while lower on AIME24 and AIME26. The results support a practical efficiency/performance tradeoff; they do not demonstrate equality on every benchmark or statistical equivalence.

Table 16: Full-run Qwen3-1.7B Thinking evaluation with dense rollout versus DistillSparse.

Benchmark	Metric	Dense	DistillSparse
AIME25	Mean@16	0.4322	0.4354
AIME25	Pass@16	0.5440	0.5562
AIME26	Mean@16	0.4833	0.4562
AIME26	Pass@16	0.5995	0.5874
AIME24	Mean@16	0.5687	0.5333
AIME24	Pass@16	0.7172	0.6929
AMC22/23	Mean@4	0.8072	0.8072
AMC12	Mean@4	0.7333	0.7333

The adapter trained with DSparse0.86 also improves alignment when applied to more aggressive schedules. This motivates interpreting the adapter as learning a correction useful across the evaluated sparse configurations, rather than only at the training schedule. This interpretation is limited to the tested schedules and model.

L.2 Asynchronous integration and scale

The asynchronous experiment in Appendix I demonstrates dynamic sparsity with asynchronous RL, not DistillSparse with asynchronous RL. A possible implementation could update the adapter on the dense-policy cadence and merge it into a refreshed inference checkpoint, avoiding a requirement for an independently synchronized continuous adapter stream. This remains an integration proposal: controlled LoRA-staleness experiments and in-flight base-model/adapter refresh tests have not been performed, so sensitivity to delayed adapter updates is unknown.

The full-run result above is for 1.7B. It does not establish the LoRA rank, correction capacity, optimizer-memory cost, or distillation overhead required at 8B and larger scales. The existing overhead experiment in Appendix D uses a separate configuration and should not be treated as a full-run scaling study. Quantifying these costs across model size and training horizon remains future work.

M LOWER-TAIL DISTRIBUTION AND BETA-FIT DIAGNOSTICS

We examine the per-token acceptance-rate distribution for Qwen3-1.7B Thinking in the 20K–22K generation-length interval under different sparse KV budgets. Most tokens have acceptance rates close to one, motivating a separate analysis of the lower tail.

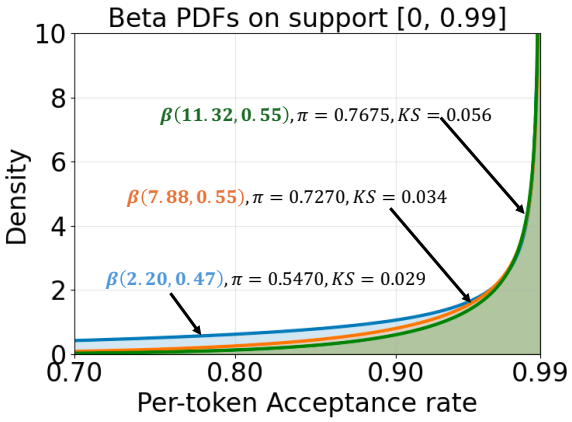

Figure 12: The tail distribution of per-token mismatch between sparse and dense can be generally approximated very well by Beta distribution. Qwen3-1.7B thinking with three sparsity (20K-22K). Budgets are 512 (blue), 8192 (coral), and 16384 (green).

Unlike the average acceptance rate, which is less informative when the distribution is highly skewed, statistics that focus on the tail distribution matter much more for evaluating the distribution mismatch between the approximate distribution and the original distribution. Notice that besides sharing similar average, the smaller KV budget distribution possesses a much thicker tail than higher KV budget. Specifically, as shown in Figure12, we find that if we leave out all the tokens with per-token acceptance rate higher than 0.999, the tail can be modeled very nicely by a scaled Beta distribution of support [0, 0.999], with all three distributions having Kolmogorov-Smirnov (KS) statistic under 0.04. From the distribution for the run with KV budget 512, the variance of the respective beta distribution is 0.19, much more spread out than 0.05 for KV budget 16384.

N ONLINE LORA DISTILLATION ALGORITHM

Algorithm 1 details the online updates used by DISTILLSPARSE. Sparse rollout trajectories and dense-policy log-probabilities are reused for distillation. The dense base model and sparse-policy adapter are updated separately: the adapter does not participate in dense-policy training, and base weights are frozen during the adapter update. Appendix D reports implementation overhead, and Appendix L provides the full-run configuration and integration limitations.

Algorithm 1 Online LoRA distillation for sparse RL training

Require: Base model parameters θ; LoRA parameters ϕ; prompt batch $x \sim \mathcal{D}$

Construct **dense policy** $\pi_\theta^{\text{dense}}$ (full attention) and **sparse policy** $\pi_{\theta,\phi}^{\text{sparse}}$ (sparse attention + LoRA)

for each RL iteration **do**

  Sample rollouts $y \sim \pi_{\theta,\phi}^{\text{sparse}}(\cdot \mid x)$ and compute rewards / advantages

  Recompute token log-probs with the dense model (FSDP): $\log \pi_\theta^{\text{dense}}(y_t \mid x, y_{<t})$

  Compute token log-probs with the sparse model: $\log \pi_{\theta,\phi}^{\text{sparse}}(y_t \mid x, y_{<t})$

  Update the base model $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{PPO}}\big(\theta; x, y, \log \pi_\theta^{\text{dense}}\big)$

  Freeze θ and update LoRA $\phi \leftarrow \phi - \eta_\phi \nabla_\phi \mathcal{L}_{\text{LoRA}}\big(\phi; x, y, \theta\big)$

  Use LoRA loss $\mathcal{L}_{\text{LoRA}} = \mathcal{L}_{\text{PPO_Sparse}} + \lambda \mathcal{L}_{\text{KL}}$

  Refresh inference engine weights with updated base model θ and LoRA ϕ

end for