SRT: Accelerating Reinforcement Learning via Speculative Rollout with Tree-Structured Cache

Chi-Chih Chang[1*] **Siqi Zhu**[2*] **Zhichen Zeng**[4] **Haibin Lin**[5] **Xin Liu**[5]
Jiaxuan You[2] **Mohamed S. Abdelfattah**[1] **Ziheng Jiang**[5] **Xuehai Qian**[3]
[1]Cornell University [2]University of Illinois Urbana-Champaign
[3]Tsinghua University [4]University of Washington [5]ByteDance

Abstract

We present Speculative Rollout with Tree-Structured Cache (SRT), a simple, model-free approach to accelerate on-policy reinforcement learning (RL) for language models without sacrificing distributional correctness. SRT exploits the empirical similarity of rollouts for the same prompt across training steps by storing previously generated continuations in a per-prompt tree-structured cache. During generation, the current policy uses this tree as the draft model for performing speculative decoding. To keep the cache fresh and improve draft model quality, SRT updates trees online from ongoing rollouts and proactively performs run-ahead generation during idle GPU bubbles. Integrated into standard RL pipelines (*e.g.*, PPO, GRPO and DAPO) and multi-turn settings, SRT consistently reduces generation and step latency and lowers per-token inference cost, achieving up to 2.08× wall-clock time speedup during rollout.

1 Introduction

Reinforcement learning (RL) has emerged as a pivotal paradigm for scaling language models, enabling them to tackle sophisticated problems like competition-level mathematics and programming tasks through deeper and longer reasoning processes [GLM et al., 2025, DeepSeek-AI et al., 2025, Seed et al., 2025, OpenAI et al., 2025, Yang et al., 2025, Kimi et al., 2025].

However, as RL training scales, the rollout generation phase has become the dominant wall-clock bottleneck, consuming over 70% (see Figure 2) of total runtime in synchronous systems [Sheng et al., 2025]. This stems from the auto-regressive, memory-bound nature of token generation and the "long-tail" distribution of response lengths, where a few lengthy rollouts stall entire batches, leaving GPUs idle and underutilized [Fu et al., 2025a, Zhong et al., 2025]. The issue is exacerbated in advanced algorithms like Group Relative Policy Optimization (GRPO) [Shao et al., 2024] and Decoupled Clip and Dynamic Sampling Policy Optimization (DAPO) [Yu et al., 2025a], which sample multiple responses per prompt to compute relative advantages or filter uninformative data, thereby amplifying generation costs.

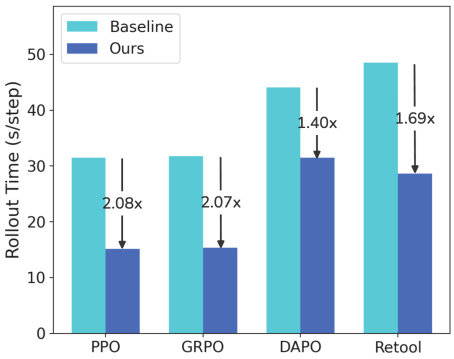

Figure 1: Rollout Speedups of SRT Across Different RL Algorithms on Qwen2.5-1.5B

[*]Equal contribution. Work during internship at Bytedance.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: The First Workshop on Efficient Reasoning.

To mitigate rollout overhead, recent algorithm-system co-design approaches [Fu et al., 2025a, Seed et al., 2025, Kimi et al., 2025] relax strict on-policy execution by asynchronously launching future rollouts without awaiting current weight updates. While these approaches boost hardware utilization and throughput by reducing idle "bubbles", they deviate from on-policy sampling, potentially introducing convergence issues in certain regimes.

In this work, we introduce *Speculative Rollout with Tree-Structured Cache* (SRT), a new acceleration paradigm that speeds up on-policy rollouts without sacrificing training efficiency. SRT leverages the observation that the policies at different training epochs exhibit a certain level of similarity when responding to the same prompt. We maintain the generated rollouts of questions and store them in a tree-structured cache, using them as a model-free draft for speculative decoding during generation: the current policy verifies and accepts drafted tokens up to the first mismatch, ensuring lossless preservation of the on-policy distribution.

To improve cache freshness and draft quality, SRT employs two complementary cache maintenance strategies. First, it inserts decoded tokens from ongoing rollouts into the tree-structured cache. Second, near the end of each step—when only a few long sequences remain—SRT leverages otherwise idle GPUs to run ahead on active prompts, generating partial rollouts that are inserted into the cache and then discarded. This preserves on-policy training: actual rollouts are produced at the designated step, and speculative continuations are never used as learning targets.

We evaluate SRT on popular RL training algorithms (PPO, GRPO, DAPO) and in the multi-turn scenario (ReTool Feng et al. [2025]). Extensive experiments show that SRT can achieve rollout speedups of up to 2.08× providing a practical, on-policy path to more efficient RL training.

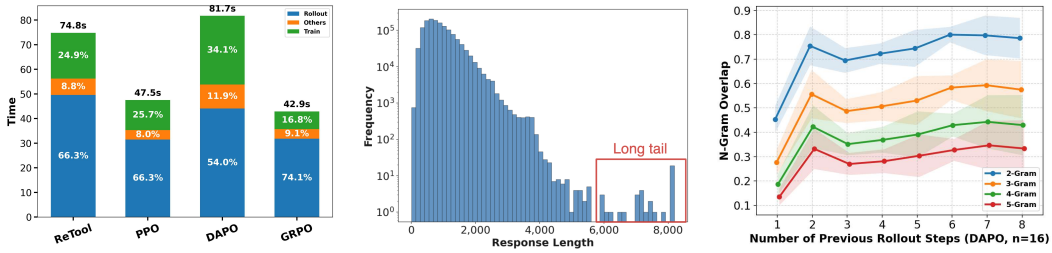

Figure 2: **(a)**. Time breakdown across different RL algorithm. **(b.)** Output length distribution on DAPO-17k dataset. **(c.)** Example of N-gram overlap for a specific prompt. The overlap is computed by comparing rollouts from the current step against the aggregated N-grams from all prior steps.[2]

2 Motivation

Dominant rollout generation time during RL training. The standard RL pipeline suffers from substantial inefficiencies, with the rollout stage emerging as the dominant computational bottleneck. As shown in Figure 2a, rollout consumes 65% of total training time across four algorithms on average, making it the most critical target for optimization.

Imbalanced response length among questions in a batch. A further structural issue arises from batched generation. Output lengths follow a long-tailed distribution (Figure 2b): while most sequences terminate quickly, a few very long responses delay batch completion. This imbalance leaves many GPUs idle, creating underutilized bubbles of compute.

Similarity of rollouts across epochs. Responses to the same question often exhibit high similarity across epochs, and this overlap grows as more rollouts are accumulated (Figure 2c). This pattern suggests an opportunity that existing RL methods fail to leverage: historical responses can be used to predict current rollouts to accelerate training. This missed opportunity wastes GPU resources on generating nearly identical content.

These factors point to a clear need for rethinking the standard RL pipeline and motivate the development of more streamlined RL strategies.

[2]Experiments are conducted using Qwen2.5-1.5B

3 Method

RL training in brief. We consider standard on-policy reinforcement learning for language models. Given a prompt $x_{1:m}$ and a policy π_θ, the learner samples one or more continuations y by autoregressively decoding from $\pi_\theta(\cdot \mid x_{1:m})$. A task-specific reward $r(x, y)$ is computed (e.g., program execution, self-consistency or preference modeling), and gradients are formed from advantages $A(x, y)$ under a clipped policy-gradient objective (PPO-style) or its multi-sample variants such as GRPO and DAPO. Training alternates between a *rollout* phase that generates K samples per prompt and an *update* phase that fits π_θ on the on-policy batch.

Per-prompt rollout cache using tree-structured cache. SRT accelerates these on-policy rollouts by maintaining, for each prompt p, a cache of previously seen token subsequences organized as a tree-structured cache $\mathcal{T}_p$. Paths in $\mathcal{T}_p$ therefore compactly index *all* substrings that have occurred in earlier generations for the same prompt, including from prior policy checkpoints. Each node corresponds to a context (a token subsequence) and stores outgoing edges labeled by next tokens, together with simple frequency statistics: $\texttt{count}(u)$ for a node u recording its frequency in previous rollouts. This structure is purely model-free and can be stored in CPU memory; it can be updated online in amortized linear time as new tokens arrive.

Speculative Rollout with Tree-Structured Cache. During rollout for prompt p, suppose we have already produced a partial continuation $y_{1:t}$. We locate the longest suffix of $y_{1:t}$ that appears in $\mathcal{T}_p$ by walking through the tree from the root along tokens $y_{t-q+1:t}$; if the walk fails, we revert to standard decoding for one step and try again. Once a match of length q is found at node u_q, SRT assembles a set $\widehat{\mathcal{T}}$ of draft tokens by greedily adding descendants of u_q that are most likely to be accepted. We rank an edge to child v by empirical conditional

$$C(v) = \frac{\texttt{count}(v)}{\sum_{w \in \texttt{children}(\texttt{parent}(v))} \texttt{count}(w)},$$

and score a node by the product of scores along its path from u_q. Intuitively, $C(v)$ estimates how often a specific next token followed this prefix in prior rollouts, and the path product estimates the chance that a drafted chain of tokens will align with what the current policy would generate. We continue expansion until a budget $B(q)$ is reached. Given $\widehat{\mathcal{T}}$, SRT performs one decode pass of the current policy to verify multiple drafted tokens in parallel following classic speculative decoding procedure [Cai et al., 2024, Miao et al., 2023, Li et al., 2024, Leviathan et al., 2023, Oliaro et al., 2025].

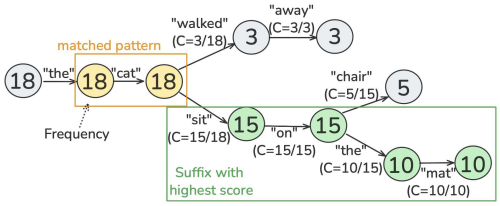

Figure 3: Given the matched prefix "the cat", we choose its suffix "sit on the mat" with highest score as draft tokens. Leaf nodes show the number of times each suffix appeared in cached rollouts, while non-leaf nodes contain the sum of their children's counts.

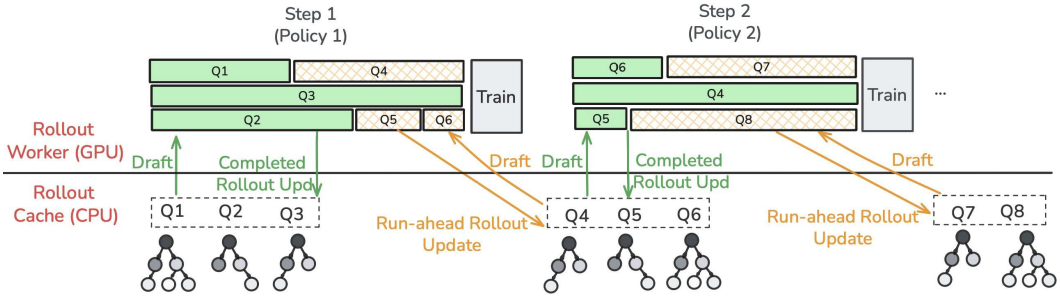

Figure 4: Illustration of cache maintenance strategy in SRT.

Cache update strategy. The maintenance of the rollout cache is crucial to the achievable speedups. In SRT, we maintain the cache by two sources as illustrated in Figure 4. First, decoded outputs of *running rollouts* are inserted online into $\mathcal{T}_p$ and node counts are updated. This immediately benefits the remaining samples for the same prompt in multi-sample algorithms and carries signal across training steps when the prompt reappears. Second, we exploit *run-ahead generation* during bubbles. Whenever some sequences in a batch finish early and GPU compute would have otherwise been idle,

we allocate that slack to generate rollouts of prompts that will be sampled soon (e.g., from the data loader's look-ahead window or an active prompt queue). Run-ahead tokens are inserted into $\mathcal{T}_p$ but are never used for learning targets; they serve solely as future drafting hints. Unlike a history-only caches design [He et al., 2025b], which caches only completed responses from previous epochs in an offline and asynchronous fashion, SRT's maintenance strategy mitigate cold-start for first-time prompts, actively enriches the cache, and yields more accepted tokens per decoding step—reducing decoding steps and end-to-end latency.

4 Experiment

4.1 End-to-end Results

Table 1: **SRT achieves superior performance over other methods.**

	Gen (s) (↓)				**Step (s) (↓)**				**μs/token (↓)**			
Method	PPO	GRPO	DAPO	Retool	PPO	GRPO	DAPO	Retool	PPO	GRPO	DAPO	Retool
Baseline	31.5	31.8	44.1	49.0	47.5	42.9	81.7	74.8	104	83.8	32.9	121
N-gram	31.4	31.1	46.0	45.0	47.3	42.1	84.5	69.3	105	82.4	33.3	161
SuffixDecoding	18.4	19.7	62.5	38.8	35.9	30.7	103	69.2	56.6	52.4	41.1	76.7
SRT (Ours)	**15.2**	**15.4**	**31.5**	**28.7**	**31.5**	**26.2**	**68.7**	**58.8**	**48.3**	**41.6**	**23.3**	**62.2**

Effectiveness of Per-prompt Rollout Cache. We present the experiment results without run-ahead generation to examine the potential of the proposed rollout cache. We integrated SRT into vLLM [Kwon et al., 2023] and used it as the inference engine for Verl [Sheng et al., 2025]. Our end-to-end experiments were conducted with the Qwen2.5-1.5B model, using on-policy RL across four algorithms. Concretely, PPO and GRPO were trained on the math dataset [Hendrycks et al., 2021], while DAPO and ReTool were trained on DAPO-Math-17k [Yu et al., 2025b]. As summarized in Table 1, SRT consistently outperforms speculative decoding strategies that were originally designed for non-RL scenario across various RL algorithms. In particular, SRT achieves lower generation and step latency as well as reduced per-token inference cost, and these gains hold robustly across both single-turn (PPO/GRPO/DAPO) and multi-turn (ReTool) training regimes.

4.2 Effect of On-the-Fly Updates and Run-Ahead Generation

In this section, we present simulation studies evaluating the impact of (i) on-the-fly updates from ongoing rollouts and (ii) run-ahead generation, both of which enrich the tree-structured cache. Using the DAPO algorithm on DAPO-17k [Yu et al., 2025a], we compare SRT to a history-only baseline that updates the cache solely with fully completed responses from previous epochs. As shown in Fig. 5, SRT consistently achieves a higher mean of accepted tokens (per decoding step). Enabling run-ahead yields additional gains, indicating that a richer cache produces higher-quality drafts. In practice, this reduces decoding steps per prompt, thereby reducing end-to-end cost and latency.

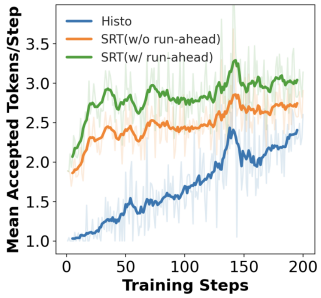

Figure 5: Mean accepted tokens analysis of different cache maintenance strategy.

5 Conclusion

We introduced Speculative Rollout with Tree-Structured Cache (SRT), a speculative rollout algorithm that accelerates on-policy reinforcement learning by exploiting redundancy across rollouts. By organizing past continuations into per-prompt tree-structured caches and leveraging speculative decoding with run-ahead generation, SRT achieves substantial reductions in generation latency and inference cost without compromising distributional correctness. Experiments across multiple RL algorithms and multi-turn settings demonstrate up to 2.08× rollout speedup, highlighting SRT as a practical framework for scaling efficient RL training.

References

T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, and T. Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads, 2024. URL https://arxiv.org/abs/2401.10774.

DeepSeek-AI, D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, X. Zhang, X. Yu, Y. Wu, Z. F. Wu, Z. Gou, Z. Shao, Z. Li, Z. Gao, A. Liu, B. Xue, B. Wang, B. Wu, B. Feng, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Ding, H. Xin, H. Gao, H. Qu, H. Li, J. Guo, J. Li, J. Wang, J. Chen, J. Yuan, J. Qiu, J. Li, J. L. Cai, J. Ni, J. Liang, J. Chen, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Zhao, L. Wang, L. Zhang, L. Xu, L. Xia, M. Zhang, M. Zhang, M. Tang, M. Li, M. Wang, M. Li, N. Tian, P. Huang, P. Zhang, Q. Wang, Q. Chen, Q. Du, R. Ge, R. Zhang, R. Pan, R. Wang, R. J. Chen, R. L. Jin, R. Chen, S. Lu, S. Zhou, S. Chen, S. Ye, S. Wang, S. Yu, S. Zhou, S. Pan, S. S. Li, S. Zhou, S. Wu, S. Ye, T. Yun, T. Pei, T. Sun, T. Wang, W. Zeng, W. Zhao, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, W. L. Xiao, W. An, X. Liu, X. Wang, X. Chen, X. Nie, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yang, X. Li, X. Su, X. Lin, X. Q. Li, X. Jin, X. Shen, X. Chen, X. Sun, X. Wang, X. Song, X. Zhou, X. Wang, X. Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. Zhang, Y. Xu, Y. Li, Y. Zhao, Y. Sun, Y. Wang, Y. Yu, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Ou, Y. Wang, Y. Gong, Y. Zou, Y. He, Y. Xiong, Y. Luo, Y. You, Y. Liu, Y. Zhou, Y. X. Zhu, Y. Xu, Y. Huang, Y. Li, Y. Zheng, Y. Zhu, Y. Ma, Y. Tang, Y. Zha, Y. Yan, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Xie, Z. Zhang, Z. Hao, Z. Ma, Z. Yan, Z. Wu, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Pan, Z. Huang, Z. Xu, Z. Zhang, and Z. Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

J. Feng, S. Huang, X. Qu, G. Zhang, Y. Qin, B. Zhong, C. Jiang, J. Chi, and W. Zhong. Retool: Reinforcement learning for strategic tool use in llms, 2025. URL https://arxiv.org/abs/2504.11536.

W. Fu, J. Gao, X. Shen, C. Zhu, Z. Mei, C. He, S. Xu, G. Wei, J. Mei, J. Wang, T. Yang, B. Yuan, and Y. Wu. Areal: A large-scale asynchronous reinforcement learning system for language reasoning, 2025a. URL https://arxiv.org/abs/2505.24298.

W. Fu, J. Gao, X. Shen, C. Zhu, Z. Mei, C. He, S. Xu, G. Wei, J. Mei, J. Wang, T. Yang, B. Yuan, and Y. Wu. Areal: A large-scale asynchronous reinforcement learning system for language reasoning, 2025b. URL https://arxiv.org/abs/2505.24298.

GLM, A. Zeng, X. Lv, Q. Zheng, Z. Hou, B. Chen, C. Xie, C. Wang, D. Yin, H. Zeng, J. Zhang, K. Wang, L. Zhong, M. Liu, R. Lu, S. Cao, X. Zhang, X. Huang, Y. Wei, Y. Cheng, Y. An, Y. Niu, Y. Wen, Y. Bai, Z. Du, Z. Wang, Z. Zhu, B. Zhang, B. Wen, B. Wu, B. Xu, C. Huang, C. Zhao, C. Cai, C. Yu, C. Li, C. Ge, C. Huang, C. Zhang, C. Xu, C. Zhu, C. Li, C. Yin, D. Lin, D. Yang, D. Jiang, D. Ai, E. Zhu, F. Wang, G. Pan, G. Wang, H. Sun, H. Li, H. Li, H. Hu, H. Zhang, H. Peng, H. Tai, H. Zhang, H. Wang, H. Yang, H. Liu, H. Zhao, H. Liu, H. Yan, H. Liu, H. Chen, J. Li, J. Zhao, J. Ren, J. Jiao, J. Zhao, J. Yan, J. Wang, J. Gui, J. Zhao, J. Liu, J. Li, J. Li, J. Lu, J. Wang, J. Yuan, J. Li, J. Du, J. Du, J. Liu, J. Zhi, J. Gao, K. Wang, L. Yang, L. Xu, L. Fan, L. Wu, L. Ding, L. Wang, M. Zhang, M. Li, M. Xu, M. Zhao, M. Zhai, P. Du, Q. Dong, S. Lei, S. Tu, S. Yang, S. Lu, S. Li, S. Li, Shuang-Li, S. Yang, S. Yi, T. Yu, W. Tian, W. Wang, W. Yu, W. L. Tam, W. Liang, W. Liu, X. Wang, X. Jia, X. Gu, X. Ling, X. Wang, X. Fan, X. Pan, X. Zhang, X. Zhang, X. Fu, X. Zhang, Y. Xu, Y. Wu, Y. Lu, Y. Wang, Y. Zhou, Y. Pan, Y. Zhang, Y. Wang, Y. Li, Y. Su, Y. Geng, Y. Zhu, Y. Yang, Y. Li, Y. Wu, Y. Li, Y. Liu, Y. Wang, Y. Li, Y. Zhang, Z. Liu, Z. Yang, Z. Zhou, Z. Qiao, Z. Feng, Z. Liu, Z. Zhang, Z. Wang, Z. Yao, Z. Wang, Z. Liu, Z. Chai, Z. Li, Z. Zhao, W. Chen, J. Zhai, B. Xu, M. Huang, H. Wang, J. Li, Y. Dong, and J. Tang. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025. URL https://arxiv.org/abs/2508.06471.

J. He, T. Li, E. Feng, D. Du, Q. Liu, T. Liu, Y. Xia, and H. Chen. History rhymes: Accelerating llm reinforcement learning with rhymerl, 2025a. URL https://arxiv.org/abs/2508.18588.

J. He, T. Li, E. Feng, D. Du, Q. Liu, T. Liu, Y. Xia, and H. Chen. History rhymes: Accelerating llm reinforcement learning with rhymerl, 2025b. URL https://arxiv.org/abs/2508.18588.

D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

J. Hu, X. Wu, W. Shen, J. K. Liu, Z. Zhu, W. Wang, S. Jiang, H. Wang, H. Chen, B. Chen, W. Fang, Xianyu, Y. Cao, H. Xu, and Y. Liu. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework, 2025. URL https://arxiv.org/abs/2405.11143.

Kimi, A. Du, B. Gao, B. Xing, C. Jiang, C. Chen, C. Li, C. Xiao, C. Du, C. Liao, C. Tang, C. Wang, D. Zhang, E. Yuan, E. Lu, F. Tang, F. Sung, G. Wei, G. Lai, H. Guo, H. Zhu, H. Ding, H. Hu, H. Yang, H. Zhang, H. Yao, H. Zhao, H. Lu, H. Li, H. Yu, H. Gao, H. Zheng, H. Yuan, J. Chen, J. Guo, J. Su, J. Wang, J. Zhao, J. Zhang, J. Liu, J. Yan, J. Wu, L. Shi, L. Ye, L. Yu, M. Dong, N. Zhang, N. Ma, Q. Pan, Q. Gong, S. Liu, S. Ma, S. Wei, S. Cao, S. Huang, T. Jiang, W. Gao, W. Xiong, W. He, W. Huang, W. Xu, W. Wu, W. He, X. Wei, X. Jia, X. Wu, X. Xu, X. Zu, X. Zhou, X. Pan, Y. Charles, Y. Li, Y. Hu, Y. Liu, Y. Chen, Y. Wang, Y. Liu, Y. Qin, Y. Liu, Y. Yang, Y. Bao, Y. Du, Y. Wu, Y. Wang, Z. Zhou, Z. Wang, Z. Li, Z. Zhu, Z. Zhang, Z. Wang, Z. Yang, Z. Huang, Z. Huang, Z. Xu, Z. Yang, and Z. Lin. Kimi k1.5: Scaling reinforcement learning with llms, 2025. URL https://arxiv.org/abs/2501.12599.

W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

Y. Leviathan, M. Kalman, and Y. Matias. Fast inference from transformers via speculative decoding. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Y. Li, F. Wei, C. Zhang, and H. Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024. doi: 10.48550/arXiv.2401.15077. URL https://arxiv.org/abs/2401.15077.

X. Miao, G. Oliaro, Z. Zhang, X. Cheng, Z. Wang, Z. Zhang, R. Y. Y. Wong, A. Zhu, L. Yang, X. Shi, C. Shi, Z. Chen, D. Arfeen, R. Abhyankar, and Z. Jia. Specinfer: Accelerating generative large language model serving with tree-based speculative inference and verification. *arXiv preprint arXiv:2305.09781*, 2023. doi: 10.48550/arXiv.2305.09781. URL https://arxiv.org/abs/2305.09781. ASPLOS 2024.

P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica. Ray: A distributed framework for emerging ai applications, 2018. URL https://arxiv.org/abs/1712.05889.

G. Oliaro, Z. Jia, D. Campos, and A. Qiao. Suffixdecoding: Extreme speculative decoding for emerging ai applications, 2025. URL https://arxiv.org/abs/2411.04975.

OpenAI, :, S. Agarwal, L. Ahmad, J. Ai, S. Altman, A. Applebaum, E. Arbus, R. K. Arora, Y. Bai, B. Baker, H. Bao, B. Barak, A. Bennett, T. Bertao, N. Brett, E. Brevdo, G. Brockman, S. Bubeck, C. Chang, K. Chen, M. Chen, E. Cheung, A. Clark, D. Cook, M. Dukhan, C. Dvorak, K. Fives, V. Fomenko, T. Garipov, K. Georgiev, M. Glaese, T. Gogineni, A. Goucher, L. Gross, K. G. Guzman, J. Hallman, J. Hehir, J. Heidecke, A. Helyar, H. Hu, R. Huet, J. Huh, S. Jain, Z. Johnson, C. Koch, I. Kofman, D. Kundel, J. Kwon, V. Kyrylov, E. Y. Le, G. Leclerc, J. P. Lennon, S. Lessans, M. Lezcano-Casado, Y. Li, Z. Li, J. Lin, J. Liss, Lily, Liu, J. Liu, K. Lu, C. Lu, Z. Martinovic, L. McCallum, J. McGrath, S. McKinney, A. McLaughlin, S. Mei, S. Mostovoy, T. Mu, G. Myles, A. Neitz, A. Nichol, J. Pachocki, A. Paino, D. Palmie, A. Pantuliano, G. Parascandolo, J. Park, L. Pathak, C. Paz, L. Peran, D. Pimenov, M. Pokrass, E. Proehl, H. Qiu, G. Raila, F. Raso, H. Ren, K. Richardson, D. Robinson, B. Rotsted, H. Salman, S. Sanjeev, M. Schwarzer, D. Sculley, H. Sikchi, K. Simon, K. Singhal, Y. Song, D. Stuckey, Z. Sun, P. Tillet, S. Toizer, F. Tsimpourlas, N. Vyas, E. Wallace, X. Wang, M. Wang, O. Watkins, K. Weil, A. Wendling, K. Whinnery, C. Whitney, H. Wong, L. Yang, Y. Yang, M. Yasunaga, K. Ying, W. Zaremba, W. Zhan, C. Zhang, B. Zhang, E. Zhang, and S. Zhao. gpt-oss-120b & gpt-oss-20b model card, 2025.

B. Seed, :, J. Chen, T. Fan, X. Liu, L. Liu, Z. Lin, M. Wang, C. Wang, X. Wei, W. Xu, Y. Yuan, Y. Yue, L. Yan, Q. Yu, X. Zuo, C. Zhang, R. Zhu, Z. An, Z. Bai, Y. Bao, X. Bin, J. Chen, F. Chen, H. Chen, R. Chen, L. Chen, Z. Chen, J. Chen, S. Chen, K. Chen, Z. Chen, J. Chen, J. Chen, J. Chi, W. Dai, N. Dai, J. Dai, S. Dou, Y. Du, Z. Du, J. Duan, C. Dun, T.-H. Fan, J. Feng, J. Feng, Z. Feng, Y. Fu, W. Fu, H. Fu, H. Ge, H. Guo, M. Han, L. Han, W. Hao, X. Hao, Q. He, J. He, F. He, W. Heng, Z. Hong, Q. Hou, L. Hu, S. Hu, N. Hu, K. Hua, Q. Huang, Z. Huang, H. Huang, Z. Huang, T. Huang, W. Huang, W. Jia, B. Jia, X. Jia, Y. Jiang, H. Jiang, Z. Jiang, K. Jiang, C. Jiang, J. Jiao, X. Jin, X. Jin, X. Lai, Z. Li, X. Li, L. Li, H. Li, Z. Li, S. Wan, Y. Wang, Y. Li, C. Li, N. Li, S. Li, X. Li, X. Li, A. Li, Y. Li, N. Liang, X. Liang, H. Lin, W. Lin, Y. Lin, Z. Liu, G. Liu, G. Liu, C. Liu, Y. Liu, G. Liu, J. Liu, C. Liu, D. Liu, K. Liu, S. Liu, Q. Liu, Y. Liu, K. Liu, G. Liu, B. Liu, R. Long, W. Lou, C. Lou, X. Luo, Y. Luo, C. Lv, H. Lv, B. Ma, Q. Ma, H. Ma, Y. Ma, J. Ma, W. Ma, T. Ma, C. Mao, Q. Min, Z. Nan, G. Ning, J. Ou, H. Pan, R. Pang, Y. Peng, T. Peng, L. Qian, L. Qian, M. Qiao, M. Qu, C. Ren, H. Ren, Y. Shan, W. Shen, K. Shen, K. Shen, G. Sheng, J. Shi, W. Shi, G. Shi, S. S. Cao, Y. Song, Z. Song, J. Su, Y. Sun, T. Sun, Z. Sun, B. Wan, Z. Wang, X. Wang, X. Wang, S. Wang, J. Wang, Q. Wang, C. Wang, S. Wang, Z. Wang, C. Wang, J. Wang, S. Wang, X. Wang, Z. Wang, Y. Wang, W. Wang, T. Wang, C. Wei, H. Wei, Z. Wei, S. Wei, Z. Wu, Y. Wu, Y. Wu, B. Wu, S. Wu, J. Wu, N. Wu, S. Wu, J. Wu, C. Xi, F. Xia, Y. Xian, L. Xiang, B. Xiang, B. Xiao, Z. Xiao, X. Xiao, Y. Xiao, C. Xin, S. Xin, Y. Xiong, J. Xu, Z. Xu, C. Xu, J. Xu, Y. Xu, W. Xu, Y. Xu, S. Xu, S. Yan, S. Yan, Q. Yang, X. Yang, T. Yang, Y. Yang, Y. Yang, X. Yang, Z. Yang, G. Yang, Y. Yang, X. Yao, B. Yi, F. Yin, J. Yin, Z. Ying, X. Yu, H. Yu, S. Yu, M. Yu, H. Yu, S. Yuan, J. Yuan, Y. Zeng, T. Zhan, Z. Zhang, Y. Zhang, M. Zhang, W. Zhang, R. Zhang, Z. Zhang, T. Zhang, X. Zhang, Z. Zhang, S. Zhang, W. Zhang, X. Zhang, Y. Zhang, Y. Zhang, G. Zhang, H. Zhang, Y. Zhang, R. Zheng, N. Zheng, Z. Zheng, Y. Zheng, C. Zheng, X. Zhi, W. Zhong, C. Zhong, Z. Zhong, B. Zhong, X. Zhou, N. Zhou, H. Zhou, H. Zhu, D. Zhu, W. Zhu, and L. Zuo. Seed1.5-thinking: Advancing superb reasoning models with reinforcement learning, 2025. URL https://arxiv.org/abs/2504.13914.

Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. K. Li, Y. Wu, and D. Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

G. Sheng, C. Zhang, Z. Ye, X. Wu, W. Zhang, R. Zhang, Y. Peng, H. Lin, and C. Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, page 1279–1297, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400711961. doi: 10.1145/3689031.3696075. URL https://doi.org/10.1145/3689031.3696075.

M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020. URL https://arxiv.org/abs/1909.08053.

A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, C. Zheng, D. Liu, F. Zhou, F. Huang, F. Hu, H. Ge, H. Wei, H. Lin, J. Tang, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Zhou, J. Lin, K. Dang, K. Bao, K. Yang, L. Yu, L. Deng, M. Li, M. Xue, M. Li, P. Zhang, P. Wang, Q. Zhu, R. Men, R. Gao, S. Liu, S. Luo, T. Li, T. Tang, W. Yin, X. Ren, X. Wang, X. Zhang, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Zhang, Y. Wan, Y. Liu, Z. Wang, Z. Cui, Z. Zhang, Z. Zhou, and Z. Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

N. Yang, T. Ge, L. Wang, B. Jiao, D. Jiang, L. Yang, R. Majumder, and F. Wei. Inference with reference: Lossless acceleration of large language models, 2023. URL https://arxiv.org/abs/2304.04487.

Q. Yu, Z. Zhang, R. Zhu, Y. Yuan, X. Zuo, Y. Yue, W. Dai, T. Fan, G. Liu, L. Liu, X. Liu, H. Lin, Z. Lin, B. Ma, G. Sheng, Y. Tong, C. Zhang, M. Zhang, W. Zhang, H. Zhu, J. Zhu, J. Chen, J. Chen, C. Wang, H. Yu, Y. Song, X. Wei, H. Zhou, J. Liu, W.-Y. Ma, Y.-Q. Zhang, L. Yan, M. Qiao, Y. Wu, and M. Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025a. URL https://arxiv.org/abs/2503.14476.

Q. Yu, Z. Zhang, R. Zhu, Y. Yuan, X. Zuo, Y. Yue, W. Dai, T. Fan, G. Liu, L. Liu, X. Liu, H. Lin, Z. Lin, B. Ma, G. Sheng, Y. Tong, C. Zhang, M. Zhang, W. Zhang, H. Zhu, J. Zhu, J. Chen, J. Chen,

C. Wang, H. Yu, Y. Song, X. Wei, H. Zhou, J. Liu, W.-Y. Ma, Y.-Q. Zhang, L. Yan, M. Qiao, Y. Wu, and M. Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025b. URL https://arxiv.org/abs/2503.14476.

Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer, A. Desmaison, C. Balioglu, P. Damania, B. Nguyen, G. Chauhan, Y. Hao, A. Mathews, and S. Li. Pytorch fsdp: Experiences on scaling fully sharded data parallel, 2023. URL https://arxiv.org/abs/2304.11277.

H. Zheng, Y. Zhou, B. R. Bartoldson, B. Kailkhura, F. Lai, J. Zhao, and B. Chen. Act only when it pays: Efficient reinforcement learning for llm reasoning via selective rollouts, 2025. URL https://arxiv.org/abs/2506.02177.

L. Zheng, L. Yin, Z. Xie, C. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez, C. Barrett, and Y. Sheng. Sglang: Efficient execution of structured language model programs, 2024. URL https://arxiv.org/abs/2312.07104.

Y. Zhong, Z. Zhang, B. Wu, S. Liu, Y. Chen, C. Wan, H. Hu, L. Xia, R. Ming, Y. Zhu, and X. Jin. Optimizing rlhf training for large language models with stage fusion, 2025. URL https://arxiv.org/abs/2409.13221.

A Experimental Setup

Table 2: Training configurations for four algorithms.

(a) PPO

Hyperparameter	Value	Hyperparameter	Value
Actor learning rate	1×10^{-6}	Critic learning rate	1×10^{-5}
Warmup ratio	0.0	Rollout temperature	1.0
KL Coefficient (β)	0.001	Train batch size	512
PPO mini batch size	128	Training steps	300
Max input length	512	Max response length	4096

(b) GRPO

Hyperparameter	Value	Hyperparameter	Value
Actor learning rate	1×10^{-6}	Response per Prompt	5
Warmup ratio	0.0	Rollout temperature	1.0
KL Coefficient (β)	0.001	Train batch size	128
PPO mini batch size	64	Training steps	300
Max input length	512	Max response length	4096

(c) DAPO

Hyperparameter	Value	Hyperparameter	Value
Actor learning rate	1×10^{-6}	Response per Prompt	16
Warmup ratio	0.0	Rollout temperature	1.0
KL Coefficient (β)	0.0	Train batch size	128
PPO mini batch size	32	Training steps	300
Max input length	2048	Max response length	8192
Clip ratio high	0.28	Clip ratio low	0.20

(d) ReTool

Hyperparameter	Value	Hyperparameter	Value
Actor learning rate	1×10^{-6}	Critic learning rate	2×10^{-6}
Warmup ratio	0.0	Rollout temperature	1.0
KL Coefficient (β)	0.001	Train batch size	512
PPO mini batch size	128	Training steps	300
Max input length	2048	Max response length	16384
Max turns	8	Clip ratio high	0.28
Clip ratio low	0.20		

RL training is conducted using Verl [Sheng et al., 2025], with vLLM [Kwon et al., 2023] serving as the inference engine. The experimental configuration is summarized in Table 2. All RL training experiments are performed on 8× NVIDIA H20 GPUs.

B Ablation Study

B.1 Responses per Prompt in GRPO

We analyzed the effect of varying the number of responses per prompt on the efficiency of SRT. As reported in Table 3, increasing n from 5 to 10 produces a greater performance improvement relative to competing algorithms. This behavior is consistent with expectation: as additional rollouts are incorporated, the resulting outputs exhibit greater similarity, thereby enabling SRT to exploit more reliable historical information during speculative decoding.

Table 3: Comparison of step time and generation time for GRPO.

	GRPO $n = 5$		**GRPO** $n = 10$	
Method	**Step** (s)	**Gen** (s)	**Step**	**Gen** (s)
Baseline	42.9	31.8	61.2	47.1
N-gram	42.1	31.1	58.8	45.3
SuffixDecoding	30.7	19.7	51.9	30.5
SRT	**26.2**	**15.4**	**41.2**	**20.4**
Improvement (%)	14.7	21.8	20.6	33.1

C Related Work

Efficient Reinforcement Learning Frameworks for LLMs. Recent open-source RL frameworks have democratized RL training [Sheng et al., 2025, Fu et al., 2025b, Hu et al., 2025]. A common design choice is to employ Ray [Moritz et al., 2018] to coordinate inference engines (e.g., vLLM [Kwon et al., 2023], SGLang [Zheng et al., 2024]) with training engines (e.g., Megatron [Shoeybi et al., 2020], FSDP [Zhao et al., 2023]). Building on this foundation, researchers have proposed various approaches to further improve efficiency. For example, GRESO [Zheng et al., 2025] accelerates training by skipping low-quality rollouts. Our closest concurrent work, RhymeRL [He et al., 2025a], explore the speculative rollouts by caching *historical* responses and updating its cache *offline* with outputs from the previous epoch asynchronously, without modifying the cache during the current batch's rollout, which can face history-scarcity cold-start. In SRT, instead of ingesting completed responses from the previous epoch, we (i) stream tokens *on-the-fly* from the *current batch's* rollouts into a per-prompt tree-structured cache, and (ii) perform run-ahead generation for the *future* batch to

proactively enrich the cache. This unique design mitigates RhymeRL's cold-start issue by enriching useful drafts immediately and continuously, increasing accepted tokens per step and reducing decoding cost, offering a complementary acceleration path for on-policy RL.

Speculative Decoding. Conventional LLM decoding requires accessing the KV cache at every step, making the process memory-bandwidth bound and limiting GPU efficiency. Speculative decoding mitigates this bottleneck by generating multiple candidate tokens in advance, either with a smaller draft model [Leviathan et al., 2023, Cai et al., 2024, Miao et al., 2023, Li et al., 2024] or through retrieval-based strategies [Oliaro et al., 2025, Yang et al., 2023]. These speculative tokens are subsequently verified by the base model, reducing KV cache accesses to a single pass while increasing computational intensity. Building upon this foundation, our work extends speculative decoding by reusing previous rollouts, thereby further improving the efficiency of RL training.