LADDERS: Length-Aware Data Distribution and Existing-Response Speculation for Fast RL Rollout Generation

Shengpeng Yin[1,2], **Hui-Ling Zhen**[3], **Xing Li**[3]
Mingxuan Yuan[3], **Zishuo Wang**[2], **Yuxin Peng**[2†]
[1]School of Electronic and Computer Engineering, Shenzhen Graduate School, Peking University
[2]Wangxuan Institute of Computer Technology, Peking University
[3]Huawei Noah's Ark Lab
[†]Corresponding author.
spyin25@stu.pku.edu.cn zhenhuiling2@huawei.com li.xing2@huawei.com
Yuan.Mingxuan@huawei.com wangzishuo@pku.edu.cn pengyuxin@pku.edu.cn[†]

Abstract

Reinforcement learning (RL) has become central to post-training large language models, but its rollout generation stage is often the dominant system bottleneck. The inefficiency comes from two sources. First, autoregressive decoding makes latency grow with response length. Second, response lengths vary widely within a batch: short sequences finish early, but the batch remains blocked by the longest sequence, leaving GPU capacity underutilized. This effect is amplified in multi-sample rollouts, where long responses can also concentrate on a few decoding workers and delay synchronization. We propose **LADDERS** (**L**ength-**A**ware **D**ata **D**istribution and **E**xisting-**R**esponse **S**peculation), a lightweight framework for accelerating RL rollout generation. LADDERS first uses hidden-state-based length prediction to group prompts with similar expected response lengths, and then applies an S-shaped allocation rule to balance multi-sample requests across workers. It further reuses responses already generated by the policy as draft continuations through prompt-specific suffix trees, enabling speculative decoding without an auxiliary draft model. Experiments on Qwen3-1.7B and Qwen3-8B show that LADDERS reduces rollout generation time by up to **56%** while preserving final task performance, and can be integrated into existing RL systems with minimal engineering effort.

1 Introduction

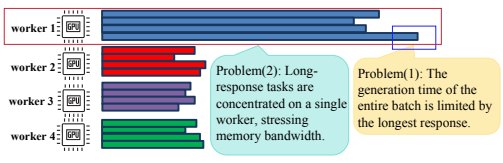


Figure 1: Two rollout-generation bottlenecks: **tail latency** within a batch and **inter-worker straggling** across decoding workers.

Large language models (LLMs) have achieved strong performance across domains [43, 24, 38, 26]. RL supports feedback control and LLM post-training for alignment, reasoning, and instruction following [30, 25, 31, 39, 34]. Modern RLHF-style pipelines usually contain three expensive stages: rollout generation, reward inference, and policy optimization [13, 32]. Among them, rollout generation is often the most costly because it repeatedly samples long responses from the current policy.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

In synchronous RLHF training, rollout generation can account for 60–70% of the iteration time [9, 35]. This bottleneck is driven by length heterogeneity. Autoregressive decoding is sequential, so per-sample latency grows roughly linearly with response length [44]. Batched decoding then inherits a longest-sequence effect: short responses may finish early, but the batch must continue until the longest response terminates, creating GPU bubbles [10, 9, 35, 41, 14] (Problem (1) in Figure 1). In multi-sample rollouts, each prompt produces multiple responses that are distributed across decoding workers. A naive average split can place several long trajectories on the same worker, causing significant increase in KV cache memory pressure on this worker (Problem (2) in Figure 1). Under heavy-tailed response lengths—studies have shown that, when the 95th-percentile length is nearly $4\times$ the median [35]—a small number of long responses can dominate wall-clock time and reduce hardware utilization.

Existing efficiency techniques [44, 37] only partially address this issue. High-throughput inference systems such as vLLM with PagedAttention [15] and kernel-level optimizations such as FlashAttention [7, 6] improve per-token efficiency, but do not remove the longest-sequence effect within a batch. Speculative decoding [17, 40, 33, 11, 23] can reduce decoding latency, but many methods require an auxiliary draft model or model-specific decoding heads [18, 19, 20, 2]. These additions increase memory cost and complicate RL training, where the target policy changes continuously. This motivates a rollout-specific framework that directly reduces length-induced straggling while avoiding major system redesign.

We propose **LADDERS** (**L**ength-**A**ware **D**ata **D**istribution and **E**xisting-**R**esponse **S**peculation), a practical framework for accelerating RL rollout generation. LADDERS combines two complementary ideas: length-aware data distribution, which reduces straggling from response-length imbalance, and existing-response speculation, which reduces effective decoding cost without an external draft model.

The first component is **length-aware data distribution** (Figure 2, left). Inspired by recent observations that prompt difficulty can be probed from LLM hidden states [16], we test whether response length can also be predicted from policy-model representations. Our results show that it can: across datasets, predicted lengths achieve Spearman rank correlations above 0.7 with ground-truth response lengths. LADDERS therefore trains a lightweight predictor on the final hidden state of the last prompt token after prefill. The predictor is used before training to assign each prompt an expected response length; from the second epoch onward, the system can replace predictions with observed rollout lengths from the previous epoch.

LADDERS uses these lengths in two scheduling steps. It first forms batches by sorting prompts according to predicted length, which reduces within-batch length variance and the resulting GPU bubbles. It then applies a length-aware S-shaped allocation rule for multi-sample rollouts. By assigning length-ordered requests to workers in alternating directions, this rule spreads long and short requests more evenly and reduces worker-level straggling.

The second component is **existing-response speculation** (Figure 2, right). Instead of training or serving an auxiliary draft model, LADDERS constructs draft continuations from responses already generated for the same prompt. Rollout generation is organized into groups. The first group is sampled normally and used to build a prompt-specific suffix tree. Later groups retrieve draft tokens from this suffix tree and verify them under the current target policy. The suffix tree can also be updated with newly generated responses and reused across epochs as a proposal source. Because accepted tokens are verified and rejected tokens are corrected under the target policy, the method preserves the target sampling distribution under the standard speculative-sampling correction while improving decoding efficiency.

Our contributions are summarized as follows: (1) We identify response-length heterogeneity as a major cause of straggling in RL rollout generation, and propose a **length-aware data distribution** strategy that predicts response lengths from policy hidden states to reduce within-batch variance and tail latency. (2) We introduce **existing-response speculation**. By organizing the decoding order into a specific two-stage paradigm, the second stage can leverage the suffix trees accumulated from the first stage and previous epochs to obtain draft tokens for accelerating speculative decoding. (3) We combine the two components into **LADDERS**, a lightweight framework that improves rollout generation speed by up to 56% while preserving downstream performance and can be implemented with only plug-in level lightweight integration.

2 Related work

2.1 Long-tail rollout generation in RL

Long-tail response lengths are a major source of inefficiency in synchronous RL post-training: a few long rollouts can dominate batch latency and leave other GPU workers idle. Recent systems mitigate this problem with length-aware scheduling or load balancing. RollPacker [9] separates predicted long generations into dedicated cycles. SortedRL [41] prioritizes short trajectories with online scheduling. RhymeRL [10] uses historical length stability for prompt-group scheduling and migration. Seer [27] partitions long rollouts with context-aware scheduling. These methods reduce straggling, but they often rely on historical statistics, pipeline-level redesign, or more complex scheduling mechanisms. LADDERS instead uses a lightweight hidden-state length predictor and an S-shaped allocation rule, making it effective from the first epoch and easy to integrate into standard rollout pipelines.

2.2 Speculative decoding for RL generation

Speculative decoding has also been explored for accelerating RL generation. Some methods exploit trajectory similarity across iterations, such as RhymeRL [10] and SPEC-RL [21]. Others introduce or adapt draft models during training, including ReSpec [3], FastGRPO [42], and RLHFSpec [35]. Although effective, these approaches often depend on auxiliary draft models, historical rollouts, or additional synchronization and tuning, which increases deployment complexity and can weaken early-stage acceleration. LADDERS organizes the generation steps into a two-stage process, where the second stage can leverage the suffix trees accumulated from the first stage and previous epochs to obtain drafts for acceleration. This design avoids external draft models, supports cold-start acceleration, and preserves the rollout sampling distribution.

3 Method

We consider the rollout generation stage in RL post-training. Given a set of prompts, the policy LLM π_θ samples variable-length responses. The bottleneck is length-induced straggling: decoding is autoregressive and often step-synchronous within a batch, so batch time is governed by the longest response while shorter responses leave compute underutilized. LADDERS addresses this bottleneck with two orthogonal components: (i) length-aware data distribution, which reduces length dispersion within batches and across workers (Figure 2 Left), and (ii) existing-response speculation, which accelerates decoding by reusing responses already produced by the policy (Figure 2 Right).

3.1 Execution model for straggling in batched decoding

Consider a rollout decoded in batches of size B. For batch b, let the generated token lengths be $L_{b,1}, \ldots, L_{b,B} \in \mathbb{N}$. In standard autoregressive decoding, one decode step is executed per token position. Ignoring prefill and assuming an approximately constant per-step cost $c > 0$, the wall-clock time of batch b is

$$T_b \approx c \cdot \max_{1 \le j \le B} L_{b,j}. \tag{1}$$

Useful computation is proportional to $\sum_{j=1}^{B} L_{b,j}$, which motivates the utilization proxy

$$U_b = \frac{\sum_{j=1}^{B} L_{b,j}}{B \cdot \max_j L_{b,j}} \in (0, 1]. \tag{2}$$

The proxy in Eq. (2) reaches 1 only when all sequences have equal length. For fixed total generated tokens $\sum_j L_{b,j}$, reducing $\max_j L_{b,j}$ improves both wall-clock time and utilization.

For N prompts split into $M = N/B$ batches, rollout time can be approximated as

$$T_{\text{rollout}} \approx c \sum_{b=1}^{M} \max_j L_{b,j}. \tag{3}$$

By Eq. (3), any scheduling strategy that lowers $\mathbb{E}[\max_j L_{b,j}]$ yields a proportional reduction in rollout time.

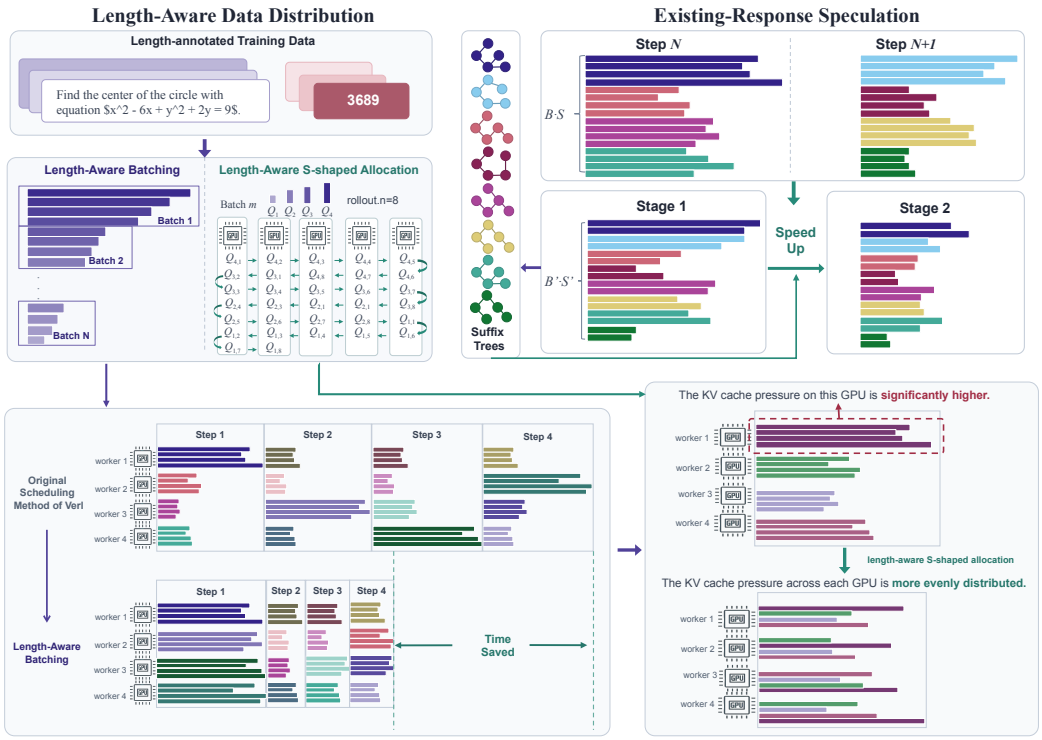


Figure 2: Overview of LADDERS. (Left) A lightweight length predictor trained on a small length-labeled subset enables length-aware batching; an S-shaped allocation rule then balances multi-sample requests across decoding workers. (Right) Existing-response speculation uses a two-stage rollout: seed responses are first generated and inserted into prompt-specific suffix trees; later responses retrieve draft continuations from these trees and verify them under the target policy, avoiding an auxiliary draft model.

3.2 Variance–time relationship: why similar lengths should be batched together

To connect length similarity with wall-clock time, we model sequence lengths in a random batch as draws with mean μ_b and variance σ_b^2, and analyze $\mathbb{E}[\max_j L_{b,j}]$. For practical length distributions with bounded budgets or light enough tails, a standard maximal inequality gives the illustrative bound

$$\mathbb{E}\left[\max_{1\le j\le B} L_{b,j}\right] \le \mu_b + \sigma_b\sqrt{2\log B}. \tag{4}$$

Substituting Eq. (4) into the batch-time model in Eq. (1) gives

$$\mathbb{E}[T_b] \lesssim c\left(\mu_b + \sigma_b\sqrt{2\log B}\right). \tag{5}$$

For a fixed mean workload, the expected straggler length grows approximately linearly with the within-batch standard deviation σ_b.

Summing Eq. (5) over all batches yields

$$\mathbb{E}[T_{\text{rollout}}] \lesssim c\sum_{b=1}^{M}\mu_b + c\sqrt{2\log B}\sum_{b=1}^{M}\sigma_b. \tag{6}$$

In Eq. (6), the first term is determined by the total token workload, while the second term captures straggling overhead from within-batch dispersion. Reducing $\sum_b \sigma_b$ therefore reduces the expected rollout time. This motivates length-aware batching: place prompts with similar expected response lengths in the same batch so that each batch has smaller dispersion and a shorter expected maximum.

3.3 Length-aware data distribution

3.3.1 Log-quantile hidden-state length predictor

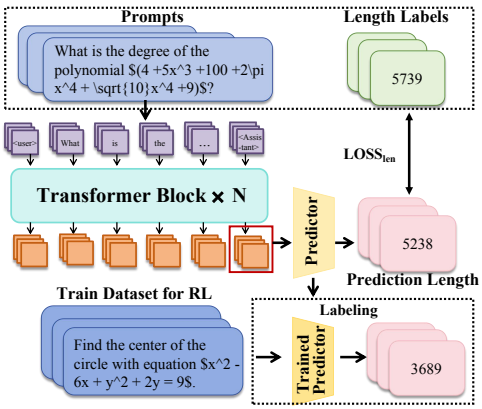


Figure 3: The length predictor maps the final prompt-token hidden state to ordered log-space length quantiles and uses the median quantile as the scheduling score.

From the analysis above, reducing sequence-length variance within batches improves throughput. To estimate the dataset's length distribution, we propose a log-quantile hidden-state length predictor (see Figure 3). For prompt $x_i = (x_{i,1}, \ldots, x_{i,m_i})$, let $h_i \triangleq H_\theta(x_i) \in \mathbb{R}^d$ be the final-layer hidden state of the last prompt token after prefill. LADDERS uses h_i as the input to a lightweight log-quantile predictor. For each prompt, the length label y_i is the median length of sampled responses. Instead of regressing y_i directly, we train in a standardized log-length space and predict three ordered conditional quantiles:

$$\begin{aligned} z_i &= \frac{\log(1+y_i) - \mu}{\sigma}, \\ (\hat{z}_{i,0.2}, \hat{z}_{i,0.5}, \hat{z}_{i,0.8}) &= \mathrm{sort}(f_\phi(h_i)), \\ \hat{\ell}_i &= \mathrm{clip}(\exp(\sigma \hat{z}_{i,0.5} + \mu) - 1, 0, L_{\max}). \end{aligned} \tag{7}$$

where μ and σ are estimated only from the predictor-training split. The log transform reduces the influence of rare very long responses, and the side quantiles regularize the median by exposing the model to conditional length uncertainty. The inverse-transformed median $\hat{\ell}_i$ in Eq. (7) is used as the scheduling score. Appendix A gives the full predictor specification.

3.3.2 Training objective

The predictor is optimized with

$$\mathcal{L}_{\text{len}} = \mathcal{L}_{\text{huber}} + 0.5\mathcal{L}_{\text{quantile}} + 0.1\mathcal{L}_{\text{rank}}. \tag{8}$$

In Eq. (8), the Huber term (Eq. (15)) is applied to the median output $\hat{z}_{i,0.5}$, the quantile term (Eq. (16)) is the pinball loss over $\{0.2, 0.5, 0.8\}$, and the ranking term (Eq. (17)) is a pairwise margin loss on median predictions. This preserves stable absolute estimates while directly encouraging the relative ordering required by Spearman-based scheduling.

3.3.3 Length-aware batching

Before rollout, each prompt obtains a predicted length $\hat{\ell}_i$. LADDERS sorts prompts by $\hat{\ell}_i$ and forms contiguous batches of size B. Let $(i_1, \ldots, i_N)$ be the sorted indices with $\hat{\ell}_{i_1} \le \cdots \le \hat{\ell}_{i_N}$. Batch b is $\{i_{(b-1)B+1}, \ldots, i_{bB}\}$. This simple sorted batching heuristic reduces within-batch length dispersion in practice. By the variance–time analysis above, smaller dispersion lowers the expected longest response in each batch and therefore reduces rollout time. (As shown in the lower-left corner of Figure 2)

3.3.4 Length-aware S-shaped allocation

Length-aware batching reduces within-batch heterogeneity, but multi-sample rollouts can still create worker imbalance if several long requests are assigned to the same worker. Consider a batch of m prompts sorted by decreasing predicted length, denoted by $Q_1, \ldots, Q_m$ with $\hat{\ell}_1 \ge \cdots \ge \hat{\ell}_m$. Each prompt Q_a requires n sampled responses, producing mn requests. With t decoding workers, the expected load of worker w is approximately

$$\mathbb{E}[\mathrm{load}(w)] \approx \sum_{(a,s)\in\mathcal{A}_w} \hat{\ell}_a, \tag{9}$$

where $\mathcal{A}_w$ is the set of requests assigned to worker w. The objective is to reduce the makespan $\max_w \mathbb{E}[\mathrm{load}(w)]$ using the loads in Eq. (9).

S-shaped assignment. LADDERS flattens requests into a length-ordered sequence indexed by $p \in \{0, \ldots, mn-1\}$, ordered first by a and then by s for requests (Q_a, s) with $a = 1, \ldots, m$ and $s = 1, \ldots, n$. The p-th request is assigned to worker $w(p)$ by alternating the traversal direction every t requests:

$$w(p) = \begin{cases} p \bmod t, & \text{if } \lfloor p/t \rfloor \text{ is even,} \\ (t-1) - (p \bmod t), & \text{if } \lfloor p/t \rfloor \text{ is odd.} \end{cases} \tag{10}$$

Equation (10) creates an S-shaped distribution over workers. (As shown in the Length-Aware S-shaped Allocation module on the left side of Figure 2)

Balancing mechanism. The reversal cancels load imbalance across adjacent rounds. For worker u, the assigned indices in consecutive rounds r and $r+1$ are $p_1 = rt+u$ and $p_2 = (r+1)t+(t-1-u)$, whose sum $p_1 + p_2 = (2r+1)t + (t-1)$ is independent of u. Since requests are sorted by predicted length, a heavier request in one round is paired with a lighter request in the next. This reduces load variance across workers. When $n = t$, each worker receives exactly one response for each prompt, giving identical expected loads under the per-prompt length estimate. In practice, S-shaped allocation spreads long-prompt samples across workers, improving synchronization and GPU utilization. (As shown in the lower-right corner of Figure 2)

3.4 Existing-response speculation with suffix trees

Length-aware data distribution reduces straggling but does not reduce per-token decoding cost. LADDERS further accelerates rollout generation with existing-response speculation. The key idea is to build drafts from responses already generated for the same prompt, rather than from an auxiliary draft model. This design provides acceleration in the first epoch because the draft repository is created within the current rollout.

3.4.1 Parallelism-preserving grouped rollout

Assume the baseline uses prompt batch size B and samples S responses per prompt, so each rollout microbatch contains $B \cdot S$ active sequences. Choose an integer expansion factor $\alpha \geq 2$ such that α divides S, and set $B' = \alpha B$ and $S' = S/\alpha$. Each grouped rollout microbatch then contains $B'S' = (\alpha B) \cdot (S/\alpha) = BS$ active sequences, preserving instantaneous decoding parallelism.

For each group of B' prompts, LADDERS partitions the S responses per prompt into α groups of size S'. Generation proceeds as follows: **Stage 1: seed responses.** Generate the first S' responses for each of the B' prompts using standard sampling from π_θ. Cache these responses as the initial prompt-specific draft repository. **Stage 2: speculated responses.** Generate the remaining $\alpha - 1$ response groups with speculative decoding. For each prompt, draft tokens are retrieved from its cached responses, verified under the target policy, and then inserted into the repository after completion.

Because the seed repository is constructed in the same rollout, LADDERS does not require previous-epoch histories to start accelerating. From the second epoch onward, suffix trees from earlier epochs can also serve as additional proposal sources; target-policy verification preserves correctness even when such proposals come from older policies.

3.4.2 Draft retrieval with a frequency-annotated suffix tree

Fix a prompt i. Let Stage 1 generate $R_i^{(1)} = \{r_{i,1}, \ldots, r_{i,S'}\}$, where each response is $r_{i,s} = (w_{i,s,1}, \ldots, w_{i,s,\ell_{i,s}})$. LADDERS builds a suffix tree, equivalently a compact trie over all suffixes of $\{r_{i,s}\}$ [36]. The tree supports the following query: given a suffix of the current generated prefix $p = (p_1, \ldots, p_t)$, return a continuation $d_{1:K}$ such that the matched suffix followed by $d_{1:K}$ forms a valid path in the tree.

To improve draft quality, each node stores a *pass count*, i.e., the number of inserted suffixes that pass through the node during construction. When multiple outgoing edges are available during draft retrieval, LADDERS greedily follows the child with the largest pass count, breaking ties arbitrarily. This biases drafts toward continuations that appear more frequently among existing responses for the same prompt. Appendix F gives the detailed matching algorithm.

3.4.3 Target-policy verification and cold-start acceleration

Given a prefix p and a draft sequence $d_{1:K}$, LADDERS verifies the draft under the current policy π_θ. The longest accepted prefix is appended to the response; at the first rejection, the next token is corrected by sampling from the target-policy distribution, and generation continues from the updated prefix. Under the standard speculative-sampling correction, this verification-and-correction step preserves the same distribution as autoregressive sampling from π_θ.

The method is cold-start free: Stage 1 creates prompt-specific draft repositories at the beginning of each rollout, and Stage 2 immediately uses them for speculation. Thus LADDERS reduces decoding cost without serving an auxiliary draft model, while keeping the generated responses on policy.

3.5 Low-effort integration of the two components

Length-aware data distribution and existing-response speculation address complementary inefficiencies. The former reduces straggling by balancing response lengths within batches and across workers; the latter reduces effective decoding cost by reusing policy-generated responses as draft proposals. Together, they mitigate both long-tail latency and per-token decoding overhead. Implementation requires only a dataloader-level sampler, an S-shaped worker assignment rule, and a speculative decoding client backed by a suffix-tree service; the policy architecture and RL objective remain unchanged.

4 Experiments

4.1 Experimental overview and setup

We evaluate LADDERS on `Qwen3-1.7B` and `Qwen3-8B` [28]. The experiments focus on two rollout bottlenecks: (i) synchronization stragglers caused by long-tailed response lengths, and (ii) throughput limits caused by autoregressive decoding. We ask four questions: (1) whether a small length-labeled subset is sufficient to train a length predictor for larger datasets; (2) whether length-aware batching and S-shaped allocation reduce intra-batch and inter-worker imbalance; (3) how existing-response speculation changes the acceptance metric τ and per-step latency during training; and (4) whether the full system reduces end-to-end rollout time without degrading final accuracy.

Hardware and training configuration. All training-related experiments are conducted on a single node with 8 GPUs, each with 32GB memory. All experiments are implemented on GRPO within the `verl` framework [32]. Unless otherwise specified, we randomly sample 1024 prompts from `MATH`, use batch size 8, set `rollout.n=8` and `TP=1`, train for 2 epochs, and log timing and system statistics. The maximum response length is 8192 tokens. For end-to-end comparisons, we report rollout wall-clock time and PASS@1 on several math benchmarks.

Baselines. For the end-to-end study, vanilla GRPO in `verl` is the primary system baseline. We additionally choose contemporaneous open-source SPEC-RL [21] as a baseline under the same model, data, and evaluation protocol.

4.2 Effectiveness of length prediction

We evaluate whether prompt hidden states can provide a reliable ordering of response lengths, which is the key signal required by length-aware scheduling. We use the log-quantile residual MLP in Appendix A: it takes the final hidden state of the last prompt token, trains in standardized log-length space, predicts the 0.2, 0.5, and 0.8 conditional quantiles, and uses the inverse-transformed median as the scheduling score. The predictor is trained on limited length-labeled subsets from `MATH` [12], `DeepScaleR` [22], and `GSM8K` [4], with two random train/test splits: 10%/90% (`10p`) and 5%/95% (`5p`). Length labels are the median lengths of three sampled responses per prompt. The policy model is frozen during hidden-state extraction, and evaluation uses Spearman rank correlation, which directly reflects the quality of the predicted ordering for scheduling.

Table 1: Spearman correlation between predicted and true response lengths under 10%/90% and 5%/95% train/test splits.

Model	Split	MATH	DeepScaleR	GSM8K
Qwen3-1.7B	10p	0.867	0.750	0.756
	5p	0.857	0.740	0.741
Qwen3-8B	10p	0.866	0.767	0.724
	5p	0.857	0.757	0.705

Table 2: Spearman correlations on additional datasets and model families.

Model	Split	MBPP	UltraFeedback	GPQA
Qwen3-8B	10p	0.730	0.825	0.686
	5p	0.709	0.808	0.678
Llama-3.1-8B-Instruct	10p	0.780	0.882	0.773
	5p	0.768	0.853	0.711

Table 1 shows that the predictor achieves strong rank correlation on the main training datasets: all Spearman coefficients are above 0.7 for both Qwen3-1.7B and Qwen3-8B under both `5p` and `10p`. The scatter plots in Appendix B further show clear monotonic trends, confirming that prompt hidden states encode useful response-length information. Thus, even a small labeled subset is sufficient for scheduling-oriented length prediction; Appendix D shows that hidden-state extraction, predictor training, and full-dataset labeling introduce only modest preprocessing overhead. Table 2 further evaluates transfer across prompt distributions (covering code, daily conversations, and scientific question answering) and model families, including `MBPP` [1], `UltraFeedback` [5], `GPQA` [29], and `Llama-3.1-8B-Instruct` [8]. On `MBPP`, `UltraFeedback`, and `GPQA`, correlations range from 0.678 to 0.882, with the strongest results on `UltraFeedback` and generally stable performance on `MBPP` and `GPQA`. Increasing the labeled-data ratio from `5p` to `10p` consistently improves performance for both Qwen3-8B and Llama-3.1-8B-Instruct. These results indicate that length prediction is a practical scheduling signal rather than a dataset-specific heuristic. Unless otherwise specified, subsequent experiments use predictor-based length estimates from the `5p` split.

4.3 Effectiveness of length-aware data distribution

4.3.1 S-shaped allocation

Multi-sample rollouts can still suffer from stragglers if several long samples are assigned to the same worker. S-shaped allocation sorts requests by predicted length and maps them to workers in an alternating back-and-forth order, so long and short requests compensate each other across workers. We use $t = 8$ decoding workers, one per GPU, and test three (m, n) settings: $(32, 16)$, $(128, 4)$, and $(64, 8)$. Prompts are randomly sampled from `MATH`. We compare average split with S-shaped split and measure one-pass generation time.



Figure 4: Token load across decoding workers under different (m, n) settings for Qwen3-1.7B and Qwen3-8B, comparing average split (AS) with S-shaped split (SS).

Results. Table 3 shows that S-shaped allocation consistently reduces generation time across both models and all settings. For Qwen3-1.7B with $(32, 16)$, generation time decreases from 181s to 122s. For Qwen3-8B under the same setting, it decreases from 1280s to 758s. Figure 4 reports total generated tokens per worker as a proxy for decoding load. The S-shaped split produces more balanced worker loads, whereas average split can concentrate long-response tasks on a few workers. These results match the load-balancing mechanism described in Section 3.3.4.

Table 3: Generation time in seconds under different (m, n) configurations, comparing average split with S-shaped split.

Model	Setting	Average Split	S-shaped Split
Qwen3-1.7B	$m = 32, n = 16$	181s	122s(↓33%)
	$m = 128, n = 4$	159s	127s(↓20%)
	$m = 64, n = 8$	140s	112s(↓20%)
Qwen3-8B	$m = 32, n = 16$	1280s	758s(↓41%)
	$m = 128, n = 4$	994s	788s(↓21%)
	$m = 64, n = 8$	738s	676s(↓8%)

4.3.2 Length-aware batching

Section 3.2 suggests that batch time is governed by the longest sequence and that larger within-batch length dispersion increases straggling. Length-aware batching therefore sorts prompts by predicted length and forms batches from adjacent prompts, reducing the expected maximum length in each batch. During training, we compute the standard deviation of response lengths within each rollout step and report its average over steps. We compare random batching with length-aware batching.

Results. As shown in Table 4, length-aware batching substantially reduces length dispersion. For Qwen3-1.7B, the average standard deviation decreases from 2320.90 to 1022.01. For Qwen3-8B, it decreases from 2302.01 to 764.80. This reduction mitigates stragglers and GPU bubbles. The resulting step-time curve becomes more structured because sorted batches separate short and long buckets, but total wall-clock time decreases. Appendix C provides detailed per-step diagnostics.

Table 4: Average within-step response-length standard deviation (↓) for random batching and length-aware batching.

MODEL	RANDOM	LENGTH-AWARE
QWEN3-1.7B	2320.90	1022.01
QWEN3-8B	2302.01	764.80

Table 5: Mean acceptance length τ and time per step for Qwen3-1.7B and Qwen3-8B across rollout stages. (Here, 1-2 denotes Epoch 1, Stage 2, etc.)

MODEL	METRIC	1-2	2-1	2-2
QWEN3-1.7B	τ(↑)	0.785	0.987	1.092
	TIME/STEP(↓)	55	51	49
QWEN3-8B	τ(↑)	0.786	0.981	1.084
	TIME/STEP(↓)	155	145	141

4.4 Speculative decoding acceptance metric

We evaluate existing-response speculation across training stages. The method reuses policy-generated responses as suffix-tree drafts and verifies each draft under the target policy. We report τ, the average number of accepted draft tokens per speculative verification step.

Results. Table 5 shows that τ increases as training proceeds, while time per step decreases. For Qwen3-1.7B, time per step drops from 55s to 49s; for Qwen3-8B, it drops from 155s to 141s. The main reason is that the suffix-tree repository grows over time and can be reused across epochs, increasing the chance of matching useful continuations and accepting longer draft prefixes. Appendix E shows the per-step time variation during speculative decoding.

4.5 Suffix-tree service overhead

We benchmark the production C++ tree server and shared-memory client using stored Qwen3-1.7B/8B seed responses, four responses per prompt, and $K = 4$, $L_{\min} = 3$, $L_{\max} = 7$. Each construction setting has 10 independent trials. Table 6 reports client-observed construction latency, including serialization and shared-memory transfer, and incremental resident proportional set size (PSS).

Table 6: Suffix-tree construction and memory overhead. Values are means; brackets give 95% confidence intervals for the 16-prompt setting.

	Build (ms)			Incremental PSS (MiB)		
Model	1 prompt	8 prompts	16 prompts	1 prompt	8 prompts	16 prompts
Qwen3-1.7B	28.6	100.9	201.1 [190.4, 211.6]	5.12	40.54	86.68 [80.11, 92.78]
Qwen3-8B	22.7	113.9	204.0 [194.0, 213.6]	4.19	45.19	86.32 [79.84, 92.56]

After warm-up, 1,000 lookup calls on 16-prompt trees take on average 77.9 μs/0.403 ms/3.161 ms for 1/8/64 queries per call on 1.7B, and 76.7 μs/0.431 ms/3.137 ms on 8B. Eight-query P95 latencies are 0.451/0.630 ms. Deletion takes 161.6/164.0 ms; construction plus deletion is 362.7/367.9 ms, or 0.74%/0.26% of the corresponding 49 s/141 s rollout steps. Memory is approximately 0.33 KiB per cached token. Lookup recurs during decoding; Appendix G additionally measures the complete online path, including lookup and verification.

4.6 End-to-end comparison and ablations

We compare LADDERS with vanilla GRPO in `verl`, the contemporaneous open-source work SPEC-RL [21], and two ablations of our method: length-aware scheduling (LS) and existing-response speculative decoding (SD).

Table 7: Rollout generation time and Pass@1 on math benchmarks for vanilla GRPO-verl, SPEC-RL, the LADDERS ablations, and full LADDERS on Qwen3-1.7B and Qwen3-8B.

Model	Method	Gen Time↓	MATH500↑	AIME24↑	AIME25↑	AMC23↑
Qwen3-1.7B	verl	4.70h	0.764	0.200	0.167	0.550
	SPEC-RL	4.11h(↓13%)	0.768	0.167	0.267	0.600
	verl+LS	3.11h(↓34%)	0.794	0.167	0.167	0.550
	verl+SD	3.87h(↓18%)	0.772	0.200	0.233	0.625
	LADDERS	**2.84h**(↓40%)	0.780	0.167	0.267	0.575
Qwen3-8B	verl	16.88h	0.808	0.233	0.267	0.600
	SPEC-RL	13.67h(↓19%)	0.802	0.367	0.167	0.700
	verl+LS	8.91h(↓47%)	0.818	0.400	0.200	0.725
	verl+SD	11.28h(↓33%)	0.822	0.433	0.233	0.650
	LADDERS	**7.42h**(↓56%)	0.806	0.400	0.267	0.650

Rollout time. Table 7 shows that SPEC-RL provides moderate speedups over `verl`, reducing generation time by 13% on Qwen3-1.7B and 19% on Qwen3-8B. LADDERS achieves larger reductions, from 4.70h to 2.84h on Qwen3-1.7B and from 16.88h to 7.42h on Qwen3-8B, corresponding to 40% and 56% reductions. The ablations show that both components contribute: LS alone reduces time by 34%/47% on Qwen3-1.7B/8B, while SD alone reduces time by 18%/33%. LS primarily improves synchronization by mitigating stragglers, and SD reduces effective decoding work; their combination gives the largest speedup.

Final accuracy. On `MATH500`, `AIME24`, `AIME25`, and `AMC23`, LADDERS achieves Pass@1 comparable to both `verl` and SPEC-RL, with small gains on some benchmarks. This indicates that LADDERS provides system-level acceleration: it changes rollout organization and decoding proposals, but does not add an auxiliary draft model or change the target policy distribution.

5 Conclusion

We identify two major efficiency bottlenecks in RL-based LLM post-training: autoregressive decoding cost and length-induced straggling. LADDERS addresses both with a lightweight combination of length-aware data distribution and existing-response speculation. It groups prompts with similar predicted lengths, balances multi-sample requests through S-shaped allocation, and reuses policy-generated responses through suffix-tree-based speculative decoding. Experiments on Qwen3-1.7B and Qwen3-8B show up to 56% reduction in rollout generation time while preserving final task performance. The idea behind LADDERS is lightweight at the plugin level, enabling non-intrusive integration into existing RL frameworks with minimal effort.

Acknowledgments and Disclosure of Funding

This work was supported by the grants from Beijing Natural Science Foundation (L247006) and the National Natural Science Foundation of China (62525201, 62132001, 62432001). The authors declare no competing interests.

References

[1] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[2] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of*

the 41st International Conference on Machine Learning, volume 235 of *Proceedings of Machine Learning Research*, pages 5209–5235. PMLR, 2024. URL https://proceedings.mlr.press/v235/cai24b.html.

[3] Qiaoling Chen, Zijun Liu, Peng Sun, Shenggui Li, Guoteng Wang, Ziming Liu, Yonggang Wen, Siyuan Feng, and Tianwei Zhang. ReSpec: Towards optimizing speculative decoding in reinforcement learning systems. *arXiv preprint arXiv:2510.26475*, 2025.

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

[5] Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, Zhiyuan Liu, and Maosong Sun. UltraFeedback: Boosting language models with scaled AI feedback. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 9722–9744. PMLR, 2024. URL https://proceedings.mlr.press/v235/cui24f.html. arXiv:2310.01377.

[6] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[7] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pages 16344–16359, 2022. doi: 10.52202/068431-1189.

[8] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783v1*, 2024.

[9] Wei Gao, Yuheng Zhao, Dakai An, Tianyuan Wu, Lunxi Cao, Shaopan Xiong, Ju Huang, Weixun Wang, Siran Yang, Wenbo Su, et al. RollPacker: Mitigating long-tail rollouts for fast, synchronous RL post-training. *arXiv preprint arXiv:2509.21009*, 2025.

[10] 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. *arXiv preprint arXiv:2508.18588*, 2025.

[11] Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and Di He. REST: Retrieval-based speculative decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1582–1595, 2024.

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

[13] Jian Hu, Xibin Wu, Wei Shen, Jason Klein Liu, Weixun Wang, Songlin Jiang, Haoran Wang, Hao Chen, Bin Chen, Wenkai Fang, et al. OpenRLHF: A Ray-based easy-to-use, scalable and high-performance RLHF framework. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 656–666, 2025.

[14] Qinghao Hu, Shang Yang, Junxian Guo, Xiaozhe Yao, Yujun Lin, Yuxian Gu, Han Cai, Chuang Gan, Ana Klimovic, and Song Han. Taming the long-tail: Efficient reasoning RL training with adaptive drafter. *arXiv preprint arXiv:2511.16665*, 2025.

[15] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023. doi: 10.1145/3600006.3613165.

[16] Sunbowen Lee, Qingyu Yin, Chak Tou Leong, Jialiang Zhang, Yicheng Gong, Shiwen Ni, Min Yang, and Xiaoyu Shen. Probing the difficulty perception mechanism of large language models. *arXiv preprint arXiv:2510.05969*, 2025.

[17] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR, 2023.

[18] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 28935–28948. PMLR, 2024. URL https://proceedings.mlr.press/v235/li24bt.html.

[19] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7421–7432, 2024. doi: 10.18653/v1/2024.emnlp-main.422.

[20] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *Advances in Neural Information Processing Systems*, volume 38, pages 136737–136756, 2025. doi: 10.52202/085713-4562.

[21] 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. *arXiv preprint arXiv:2509.23232*, 2025.

[22] 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. Project blog, 2025. URL https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303013a4e2. Accessed November 2, 2026.

[23] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. SpecInfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 932–949, 2024.

[24] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*, 2023. doi: 10.48550/arXiv.2307.06435.

[25] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744, 2022. doi: 10.52202/068431-2011.

[26] Yuxin Peng, Zishuo Wang, Geng Li, Xiangtian Zheng, Sibo Yin, and Hulingxiao He. A survey on fine-grained multimodal large language models. *Chinese Journal of Electronics*, 35(2):771–803, 2026. doi: 10.23919/cje.2025.00.336.

[27] Ruoyu Qin, Weiran He, Weixiao Huang, Yangkun Zhang, Yikai Zhao, Bo Pang, Xinran Xu, Yingdi Shan, Yongwei Wu, and Mingxing Zhang. Seer: Online context learning for fast synchronous LLM reinforcement learning. *arXiv preprint arXiv:2511.14617*, 2025.

[28] Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

[29] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level Google-proof Q&A benchmark. In *First Conference on Language Modeling*, 2024.

[30] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[31] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[32] 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*, pages 1279–1297, 2025.

[33] Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. SpecTr: Fast speculative decoding via optimal transport. In *Advances in Neural Information Processing Systems*, volume 36, pages 30222–30242, 2023. doi: 10.52202/075280-1314.

[34] Ding Wang, Jin Ren, Haiming Huang, and Junfei Qiao. Particle swarm optimization for adaptive-critic feedback control with power system applications. *Chinese Journal of Electronics*, 34(4):1265–1274, 2025. doi: 10.23919/cje.2024.00.287.

[35] Siqi Wang, Hailong Yang, Junjie Zhu, Xuezhu Wang, Yufan Xu, and Depei Qian. RLHFSpec: Breaking the efficiency bottleneck in RLHF training via adaptive drafting. *arXiv preprint arXiv:2512.04752*, 2025.

[36] Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory (SWAT 1973)*, pages 1–11. IEEE, 1973.

[37] Linhui Xiao, Guiping Cao, Mingyue Guo, Xianchao Guan, Fan Yang, Ming Tao, Xin Li, Yuxin Peng, and Yaowei Wang. A survey on the green development of large models: From resource-efficient architectures to hardware–software co-design. *Chinese Journal of Electronics*, 35(5):1817–1840, 2026. doi: 10.23919/cje.2025.00.438.

[38] Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. A survey on multimodal large language models. *National Science Review*, 11(12):nwae403, 2024. doi: 10.1093/nsr/nwae403.

[39] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

[40] Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11263–11282, 2024.

[41] Yiqi Zhang, Huiqiang Jiang, Xufang Luo, Zhihe Yang, Chengruidong Zhang, Yifei Shen, Dongsheng Li, Yuqing Yang, Lili Qiu, and Yang You. SortedRL: Accelerating RL training for LLMs through online length-aware scheduling. In *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models*, 2025.

[42] Yizhou Zhang, Ning Lv, Teng Wang, and Jisheng Dang. FastGRPO: Accelerating policy optimization via concurrency-aware speculative decoding and online draft learning. *arXiv preprint arXiv:2509.21792*, 2025.

[43] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Zican Dong, Yupeng Hou, Beichen Zhang, Yingqian Min, Junjie Zhang, Peiyu Liu, Xiaolei Wang, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Yiwen Hu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models. *Frontiers of Computer Science*, 20(12):2012627, 2026. doi: 10.1007/s11704-026-60308-3.

[44] Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, et al. A survey on efficient inference for large language models. *arXiv preprint arXiv:2404.14294*, 2024.

A Details of the log-quantile length predictor

This appendix gives the length-prediction details used by Section 4.2. For a prompt representation $h_i \in \mathbb{R}^d$ and nonnegative length label y_i, the predictor operates in the standardized log-length space

$$\begin{aligned} z_i &= \frac{\log(1+y_i) - \mu}{\sigma}, \\ \mu &= \frac{1}{N_{\text{train}}} \sum_{i \in \mathcal{I}_{\text{train}}} \log(1+y_i), \\ \sigma &= \text{Std}\{\log(1+y_i) : i \in \mathcal{I}_{\text{train}}\} + 10^{-6}. \end{aligned} \tag{11}$$

Only the training split is used to estimate μ and σ in Eq. (11), avoiding validation or test leakage.

The predictor is a flat residual MLP. It first projects h_i into a hidden dimension with layer normalization, GELU, and dropout, then applies residual MLP blocks of the form

$$h^{(t+1)} = \text{LayerNorm}\left(h^{(t)} + g_\phi^{(t)}(h^{(t)})\right), \tag{12}$$

where $g_\phi^{(t)}$ is a two-layer feed-forward module with layer normalization, GELU, and dropout. After the blocks in Eq. (12), the output head produces three scalar values, which are sorted to prevent quantile crossing:

$$(\hat{z}_{i,0.2}, \hat{z}_{i,0.5}, \hat{z}_{i,0.8}) = \text{sort}(W_q h_i^{(T)} + b_q). \tag{13}$$

The 0.5 quantile in Eq. (13) is used as the point prediction for scheduling; the 0.2 and 0.8 quantiles serve as distributional regularizers and can be interpreted as an uncertainty interval after inverse transformation.

The full training loss is

$$\mathcal{L}_{\text{len}} = \lambda_{\text{huber}}\mathcal{L}_{\text{huber}} + \lambda_{\text{quantile}}\mathcal{L}_{\text{quantile}} + \lambda_{\text{rank}}\mathcal{L}_{\text{rank}}, \tag{14}$$

Equation (14) uses $\lambda_{\text{huber}} = 1.0$, $\lambda_{\text{quantile}} = 0.5$, and $\lambda_{\text{rank}} = 0.1$. The Huber term is applied to the median output, $m_i = \hat{z}_{i,0.5}$:

$$\mathcal{L}_{\text{huber}} = \frac{1}{B}\sum_{i=1}^{B} \phi(m_i - z_i), \quad \phi(e) = \begin{cases} \frac{1}{2}e^2, & |e| < 1, \\ |e| - \frac{1}{2}, & |e| \geq 1. \end{cases} \tag{15}$$

For quantile set $\mathcal{T} = \{0.2, 0.5, 0.8\}$, the pinball loss is

$$\mathcal{L}_{\text{quantile}} = \frac{1}{B|\mathcal{T}|}\sum_{i=1}^{B}\sum_{\tau \in \mathcal{T}} \rho_\tau(z_i - \hat{z}_{i,\tau}), \quad \rho_\tau(u) = \max((\tau-1)u, \tau u). \tag{16}$$

The ranking term is the margin loss on the median-prediction space used for scheduling:

$$\mathcal{L}_{\text{rank}} = \frac{1}{|\mathcal{P}|}\sum_{(i,j)\in\mathcal{P}} \max\left(0, \gamma - \text{sign}(z_i - z_j)(m_i - m_j)\right), \tag{17}$$

where $\mathcal{P} = \{(i,j) : z_i \neq z_j\}$ and $\gamma = 0.03$ in our implementation. After training, a new prompt is assigned

$$\hat{\ell}_i = \text{clip}\left(\exp(\sigma\hat{z}_{i,0.5} + \mu) - 1, 0, 8192\right). \tag{18}$$

The prediction in Eq. (18) is then used for sorted batching and S-shaped allocation.

B Predicted length vs. actual length

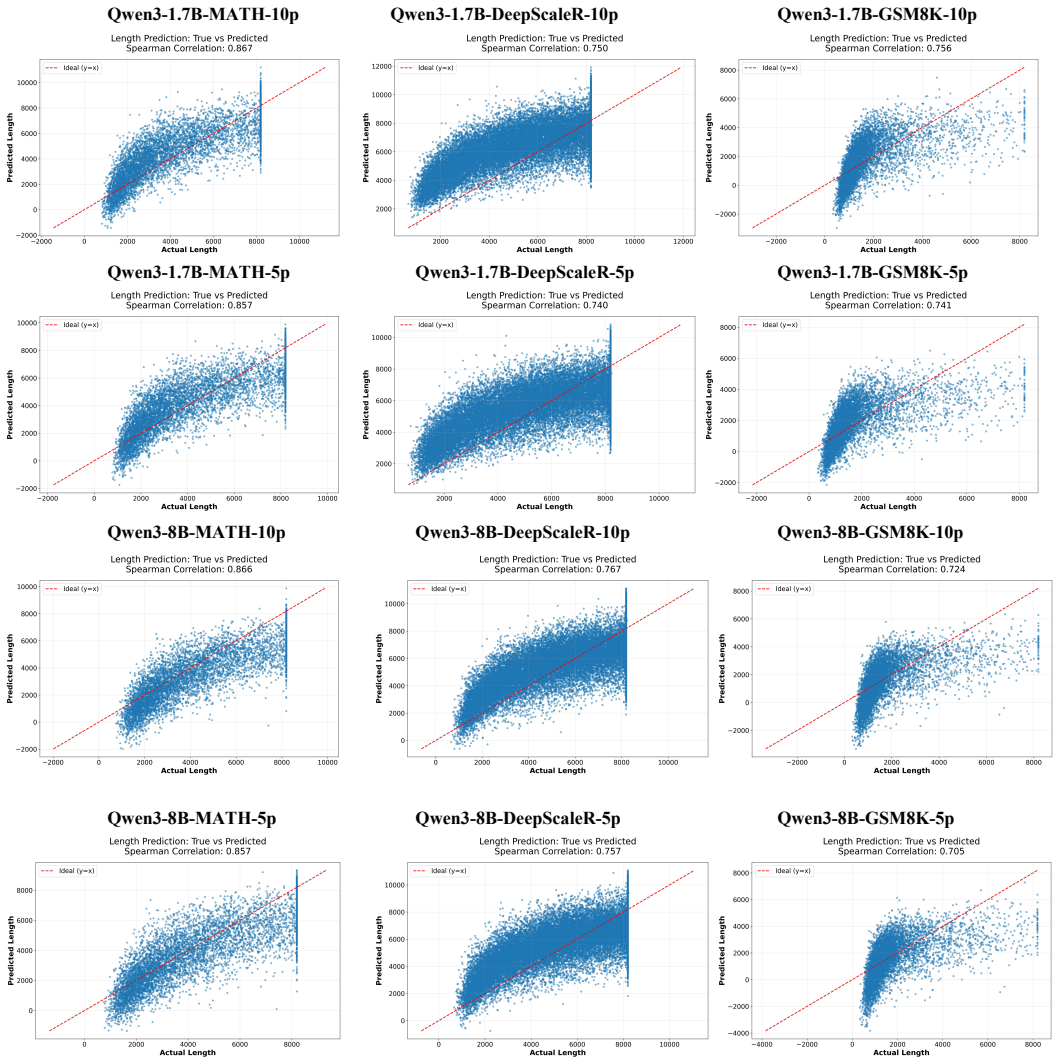


Figure 5: Scatter plots of predicted response length versus actual response length.

Figure 5 visualizes the relationship between predicted response length $\hat{\ell}$ and ground-truth length ℓ for Qwen3-1.7B and Qwen3-8B on `MATH`, `DeepScaleR`, and `GSM8K`, under the `10p` and `5p` labeled-data ratios. Each point is one prompt. The x-axis is the ground-truth length, computed as the median length over sampled responses, and the y-axis is the predictor output from prompt hidden states.

Global trend. Across all settings, the scatter plots show a clear monotonic pattern. This confirms that prompt hidden states contain response-length information. The points broadly follow the diagonal, indicating that the predictor is useful not only for ranking prompts—the main requirement for scheduling—but also for estimating typical length magnitudes.

Datasets and model sizes. The relationship is strongest on `MATH`, where both models achieve rank correlations around 0.86 under both `10p` and `5p`. Correlations are slightly lower on `DeepScaleR` and `GSM8K`, generally ranging from 0.70 to 0.77. The long-tail region has larger variance and mild saturation: very long responses are harder to predict, and dispersion grows as true length increases. This is consistent with stochastic decoding and heavy-tailed response lengths.

Supervision ratio. Reducing the labeled subset from `10p` to `5p` slightly increases scatter but preserves the overall ordering. This supports the use of a small labeled subset for training an effective length predictor.

C Per-step diagnostics for length-aware scheduling

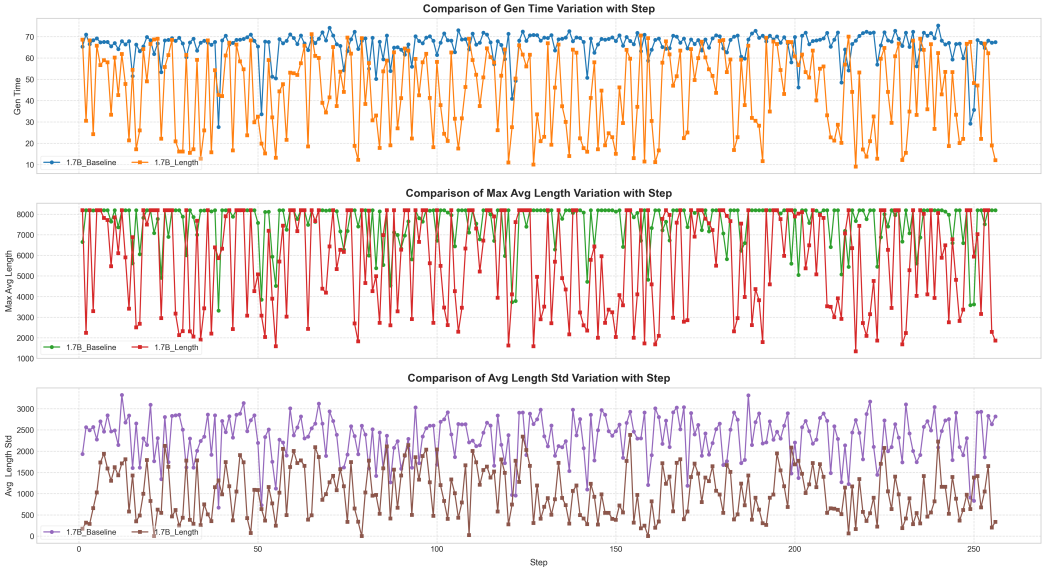


Figure 6: Per-step generation time, maximum average response length, and response-length standard deviation for random scheduling and length-aware scheduling on Qwen3-1.7B.

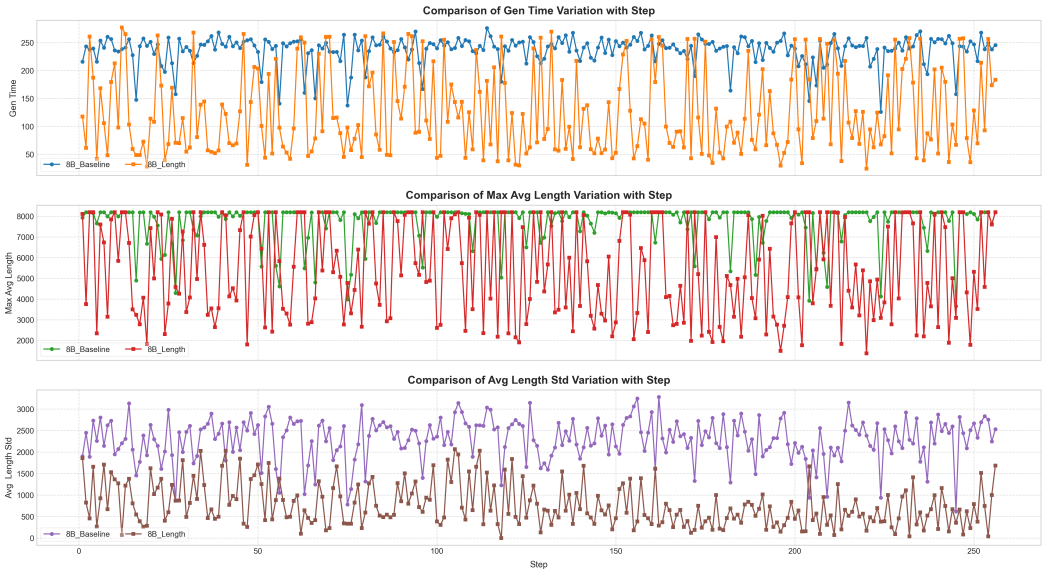


Figure 7: Per-step generation time, maximum average response length, and response-length standard deviation for random scheduling and length-aware scheduling on Qwen3-8B.

Analysis. Figures 6 and 7 provide per-step diagnostics for Qwen3-1.7B and Qwen3-8B. They compare default random scheduling with length-aware scheduling. Each figure reports (top) per-step rollout generation time, (middle) the maximum average response length within the step, used as a straggler proxy, and (bottom) the standard deviation of average response lengths across prompt groups, used as a dispersion proxy.

Step time follows the straggler length. Across both models, random scheduling produces a relatively flat but high step-time curve because random batches often mix short and very long samples. Length-aware scheduling gives much lower step times for many steps, and these reductions align with decreases in the maximum average response length. This supports the execution model that batch time is dominated by the longest sequences.

Lower dispersion reduces straggling. The bottom panels show that length-aware batching substantially reduces per-step length standard deviation, matching Table 4. By forming batches and worker assignments with more homogeneous predicted lengths, the scheduler reduces long-tail effects within each step and shrinks GPU bubbles caused by short sequences waiting for long ones.

Why the step-time curve becomes spikier. Sorting by predicted length separates the rollout into narrower length buckets. Some steps contain mostly short responses and become very fast; others contain long responses near the maximum length budget and remain slow. Although the step-time trajectory becomes more variable, overall wall-clock time decreases because most steps are no longer slowed down by accidental long responses mixed into otherwise short batches.

Consistency across model sizes. The same patterns appear on Qwen3-1.7B and Qwen3-8B: length-aware scheduling reduces the straggler proxy, lowers within-step dispersion, and reduces per-step rollout time. This indicates that the scheduling mechanism improves synchronization efficiency across model scales.

D Overhead of length prediction

Table 8: Overhead of hidden-state extraction (Forward), predictor training, and full-dataset length labeling.

		MATH		DEEPSCALER		GSM8K	
MODEL	TASK	5P	10P	5P	10P	5P	10P
QWEN3-1.7B	FORWARD	16S	22S	41S	79S	11S	22S
	TRAINING	6S	8S	14S	24S	5S	7S
	LABELING	53S		240S		36S	
QWEN3-8B	FORWARD	33S	47S	79S	150S	18S	45S
	TRAINING	7S	8S	17S	30S	4S	9S
	LABELING	161S		829S		90S	

Analysis. Table 8 reports the practical overhead of the length-prediction pipeline: extracting prompt hidden states (**Forward**), training the predictor (**Training**), and applying the predictor to label the full training set (**Labeling**). Training reports the wall-clock time for 200 predictor-training epochs on the labeled subset. Labeling reports one inference pass over the full training set.

Single-GPU timings and multi-GPU scaling. All reported times are measured on a single GPU. Forward extraction and labeling are embarrassingly parallel across samples, so using G GPUs can reduce these preprocessing costs by roughly a factor of G, up to communication and I/O overhead.

Small one-time cost. Predictor training completes within tens of seconds, and full-dataset labeling finishes within a few minutes in most settings. The largest case in Table 8 is 829s. These costs are one-time or amortized preprocessing overheads and are small compared with the end-to-end RL runtime dominated by rollout generation.

E Per-step generation time with speculative decoding

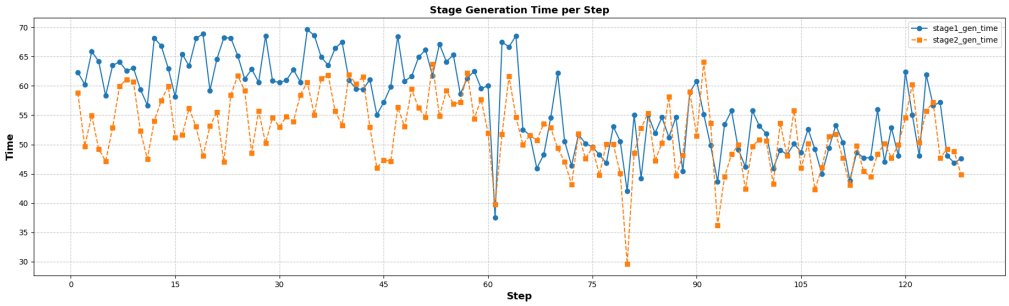


Figure 8: Per-step generation time for the two rollout stages on Qwen3-1.7B.

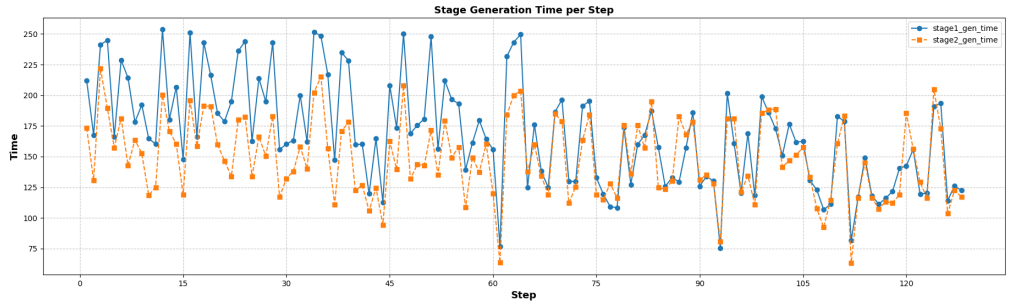


Figure 9: Per-step generation time for the two rollout stages on Qwen3-8B.

Analysis. Figures 8 and 9 report per-step generation time under the two-stage rollout with speculative decoding. Stage 1 generates seed responses and constructs prompt-specific suffix trees. Stage 2 generates later responses with suffix-tree-based draft retrieval and target-policy verification.

Stage 2 is usually faster than Stage 1. Across both Qwen3-1.7B and Qwen3-8B, the Stage 2 curve is below the Stage 1 curve for most steps. Once a suffix tree is available, draft proposals reduce the number of target-model decode iterations required per output, lowering wall-clock time per step.

The gap depends on response length and acceptance. The speedup varies across steps. Longer responses provide more opportunities to accept multi-token drafts and amortize verification overhead, producing larger gains. Short responses or lower-quality matches can make Stage 2 closer to Stage 1. This behavior is consistent with the acceptance statistics in Table 5.

Similar trend across model sizes. Qwen3-8B has higher absolute per-step time than Qwen3-1.7B because of higher per-token compute cost. Nevertheless, the same two-stage pattern appears: Stage 2 consistently reduces time and tracks Stage 1 fluctuations, suggesting that existing-response speculation remains effective at larger model scales.

F Suffix-tree speculative matching algorithm

Generalized suffix tree. A generalized suffix tree extends the classic suffix tree from one string to a set of strings. In practice, each response is appended with a distinct terminal symbol so that no suffix crosses response boundaries, and the tree is built as a compact trie over all suffixes. Leaves store response IDs and positions, while internal nodes represent substrings shared across responses. This structure supports efficient substring matching over the response set. In LADDERS, the Stage-1 responses for a prompt form the string set, and the current generated suffix is used to retrieve a plausible draft continuation.

We formalize the suffix-tree-based draft retrieval procedure. The procedure is parameterized by the maximum number of speculative tokens $K \triangleq$ `num_speculative_tokens`, the maximum suffix length $L_{\max} \triangleq$ `suffix_tree_max_prefix_len`, and the minimum suffix length $L_{\min} \triangleq$ `suffix_tree_min_prefix_len`.

Notation. Let $y_{1:t} = (y_1, \ldots, y_t)$ be the currently generated token sequence for a fixed prompt. Let $\mathcal{T}$ be the prompt-specific suffix tree built from existing responses. For any token sequence s, $\mathrm{match}(\mathcal{T}, s)$ denotes the node reached by traversing s in $\mathcal{T}$ when the path exists; otherwise, the match fails.

Goal. Given the current generated sequence $y_{1:t}$, retrieve a draft continuation $d_{1:k}$ with $0 \le k \le K$ from $\mathcal{T}$. If no valid match exists, return an empty draft.

Matching-and-drafting procedure. At each decoding iteration:

1. **Early exit for short prefixes.** If $t < L_{\min}$, return $d = \emptyset$.
2. **Bounded suffix search.** Set $\ell \triangleq \min\{t,\, L_{\max}\}$. Attempt to match the suffix $y_{t-\ell+1:t}$ in $\mathcal{T}$. If the match fails, decrement $\ell \leftarrow \ell - 1$ and retry until either a match succeeds or all lengths down to $L_{\min}$ have failed.
3. **No-match case.** If no match is found for any $\ell \in \{L_{\min}, \ldots, \min(t, L_{\max})\}$, return $d = \emptyset$.
4. **Draft retrieval.** If a match is found at node $v = \mathrm{match}(\mathcal{T}, y_{t-\ell+1:t})$, walk forward from v for at most K steps:
   (a) Initialize $k \leftarrow 0$ and $d \leftarrow ()$.
   (b) While $k < K$ and v has at least one outgoing edge, select the next child according to the tree's tie-breaking rule, such as maximum pass count; append the edge token to d; update v to the child node; and set $k \leftarrow k + 1$.
   (c) Return $d_{1:k}$.

Remarks. The procedure waits until the generated sequence reaches length $L_{\min}$ before querying the tree. It then searches from the longest allowed suffix down to the minimum suffix length, which increases the chance of finding a match while keeping query cost bounded. The returned draft has length at most K and may be shorter if the matched subtree does not contain enough continuation tokens.

G Existing-response speculation on instruction-following prompts

We evaluate less-templated instruction-following responses on `UltraFeedback` using two complementary protocols: teacher-forced trace replay and fresh online generation with the target-policy verifier. Replay agreement is measured against a fixed realized continuation and is distinct from stochastic speculative-sampling acceptance.

G.1 Full-dataset trace replay

We use all 1,967 UltraFeedback prompts, with one of three Qwen3-8B responses building the tree and two held out. This yields 3,934 target traces and 7.14M target tokens. The matched `MATH` control uses the same one-seed/two-held-out budget; we also report the paper-style four-seed/four-held-out MATH setting. Confidence intervals are 95% prompt-bootstrap intervals from 2,000 replicates.

Table 9: Teacher-forced trace replay. Nonempty draft rate is per eligible iteration; agreement τ is the number of agreed tokens per nonempty draft. Brackets denote 95% confidence intervals. Iteration compression is a replay proxy.

Dataset seed/held out	Nonempty drafts (%)	Agreement τ	Draft-token agreement (%)	Iteration compression
UltraFeedback 1/2	21.6 [21.0, 22.2]	1.265 [1.243, 1.288]	31.6	1.272× [1.263, 1.282]
MATH 1/2	37.0 [36.5, 37.4]	1.218 [1.199, 1.237]	30.5	1.449× [1.440, 1.459]
MATH 4/4	55.9 [55.5, 56.3]	1.406 [1.387, 1.425]	35.2	1.784× [1.772, 1.796]

UltraFeedback responses are more diverse: their 4-gram Jaccard overlap and distinct-4 are 0.122/0.692, versus 0.167/0.392 for MATH. As Table 9 shows, this diversity mainly lowers how often a suffix is found. Conditional agreement remains useful, with $\tau = 1.265$ and 31.6% draft-token agreement in UltraFeedback, compared with 1.218 and 30.5% in the matched MATH control.

G.2 Fresh online checkpoint run

We sample 32 UltraFeedback prompts and run the merged Qwen3-8B LADDERS checkpoint at step 128 on eight RTX 5090 GPUs. Stage 1 generates four seed responses per prompt; Stage 2 generates four fresh responses after tree construction, for 128 seed and 128 target responses totaling 461,219 tokens. The production C++ tree service and vLLM verifier use temperature 1.0, top-p 1.0, a 4,096-token budget, and $K = 4$, $L_{\min} = 3$, $L_{\max} = 7$.

Table 10: True online counters for the UltraFeedback checkpoint run. Acceptance here is measured by target-policy verification, rather than fixed-trace agreement.

Stage-2 statistic	Result
Output tokens / eligible iterations	236,927 / 166,469
Nonempty verifications / match rate	59,792 / 35.9%
Draft tokens proposed / accepted	238,953 / 70,074
Accepted tokens per nonempty verification, τ	1.172
Acceptance at positions 1/2/3/4	50.9% / 30.7% / 20.6% / 15.0%
Accepted tokens / output tokens	29.6%
Iteration compression	1.420×

All eight worker shards have $\tau > 1$ (1.061–1.355), with match rates of 26.9–46.2%. The largest per-worker tree construction takes 40.8 ms. Stage-2 throughput, including lookup and verification, is 651.0 tokens/GPU-second versus 563.6 in Stage 1, a descriptive increase of 15.5%, despite producing 5.6% more tokens. Because the stages contain independent samples, match and acceptance are the primary evidence; the throughput comparison is a consistency check rather than a randomized training-speed experiment.

H Online and offline sources of suffix-tree responses

We isolate the role of response source and prompt identity using all 1,024 MATH prompts with Qwen3-8B. Four held-out current-policy responses per prompt form a fixed target set of 4,096 traces and 17.77M tokens. For each prompt, three tree-building responses come from one of four sources: the online current policy on the same prompt, the offline base policy on the same prompt, the offline base policy on a different MATH prompt, or external DeepScaleR synthetic reasoning.

All sources are proportionally truncated to the same per-prompt token budget (mean 7,996; median 6,742), with $K = 4$, $L_{\min} = 3$, $L_{\max} = 7$. A draft agrees only when it matches the next fixed target token. The resulting target-iteration reduction measures reuse under trace replay; it is neither a wall-clock speedup nor stochastic vLLM acceptance.

Table 11: Suffix-source ablation with matched cached-token budgets. Agreement τ is agreed tokens per nonempty draft. Brackets give 95% confidence intervals for target-iteration reduction.

Tree source	Nonempty drafts (%)	Agreement τ	Target-iteration reduction (%)
Online, same prompt	40.85	1.281	34.32 [33.71, 34.93]
Offline base policy, same prompt	39.77	1.334	34.64 [34.02, 35.23]
Offline base policy, different prompt	11.86	0.253	2.91 [2.82, 3.01]
External synthetic reasoning	10.84	0.236	2.49 [2.40, 2.59]

Offline same-prompt responses retain the online benefit: the difference in target-iteration reduction is +0.32 percentage points (paired-bootstrap 95% interval [+0.12, +0.51] points), with 88.5% of prompts differing by at most five points. Changing prompt identity removes over 90% of the reduction even within MATH; external reasoning has a similarly small effect. On 32 stratified prompts, replay with the production C++ tree server differs from the simulator by at most 0.00092$\times$ in iteration compression.

These results indicate that prompt-specific solution structure accounts for most reuse. Offline base-policy responses can initialize trees for known prompts, while online collection remains the adaptive default: it avoids separate preprocessing, covers unseen prompts, and tracks policy changes. In either case, the current target policy verifies the proposed draft tokens.

I Limitations, reproducibility, and responsible release

Limitations. LADDERS is designed for synchronous RL rollout generation, where response-length heterogeneity and worker-level straggling are major bottlenecks. Its gains may be smaller when response lengths are nearly uniform, when prompt hidden states provide a weak ordering signal for length, or when a training system already uses highly asynchronous scheduling. Existing-response speculation also depends on reusable local continuations: prompts whose sampled responses share little suffix structure may obtain less benefit from suffix-tree drafts. Our end-to-end experiments focus on Qwen3-1.7B and Qwen3-8B under GRPO-style post-training, while the length-prediction study covers additional datasets and model families. Extending the evaluation to larger models, longer training runs, and more diverse RL objectives is left for future work.

Reproducibility and artifacts. The method is implemented as system-level changes around the rollout pipeline: a length-prediction preprocessing stage, a sorted sampler, an S-shaped worker assignment rule, and a suffix-tree speculative decoding client. The experimental setup in Section 4 specifies the models, datasets, rollout configuration, hardware, and main baselines, while Appendices A–F give the predictor, timing, diagnostic, and matching details. The supplemental artifact includes code, configuration files, data-preparation scripts for the public datasets, and commands for reproducing the main timing and evaluation tables.

Responsible release and asset use. This work releases an efficiency method and supporting code, not a new pretrained language model, scraped dataset, or user-facing generation service. Positive impacts include reducing the compute cost and energy footprint of RL post-training and making reproducible LLM systems research more accessible. A possible negative impact is that cheaper rollout generation can also lower the cost of training more capable models, including models that may later be misused. We therefore release the implementation as research infrastructure, require users to obtain models and datasets from their original sources, and preserve the safety and usage restrictions of the underlying assets. We use public research assets, including Qwen3, Llama-3.1-8B-Instruct, `verl`, `MATH`, `DeepScaleR`, `GSM8K`, `MBPP`, `UltraFeedback`, and `GPQA`; the paper cites the corresponding creators where used. At the time of use, Qwen3 is released under Apache-2.0, Llama-3.1 under the Llama 3.1 Community License, `verl` under Apache-2.0, `MATH`, `DeepScaleR`, `GSM8K`, `UltraFeedback`, and `GPQA` under MIT, and `MBPP` dataset content under CC BY 4.0. The released artifact documents asset versions, licenses, and download/preprocessing instructions; it will not redistribute benchmark content whose terms require users to obtain it from canonical sources. No crowdsourcing, user studies, or collection of personally identifiable information were performed for this work.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the two LADDERS components, the rollout-generation scope, and the main speedup/accuracy claims. These claims are supported by the method description and by the end-to-end results and ablations in Section 4, especially Table 7.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Appendix I discusses the main limitations, including the synchronous-rollout setting, dependence on predictable response lengths and reusable continuations, and the current model/training-regime scope.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The paper does not claim standalone formal theorems; its analytical component is a motivating execution model and variance–time relationship. The assumptions behind this analysis are stated in Sections 3.2 and the preceding execution-model subsection, and all equations are used to justify the scheduling design rather than to assert an unproved theorem.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 4 specifies the models, datasets, hardware, rollout settings, baselines, evaluation metrics, and end-to-end protocol, while the appendices give the predictor, preprocessing, diagnostic, and suffix-tree details needed to reproduce the main claims.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The supplemental artifact includes the implementation, configuration files, public-data preparation scripts, and commands for reproducing the main timing and evaluation tables. Appendix I describes the artifact and notes that datasets and models should be obtained from their original public sources.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental overview gives the GRPO/`verl` setting, model sizes, hardware, prompt sampling, batch size, rollout count, tensor parallelism, epochs, response-length budget, baselines, and metrics. Appendix A further specifies the length-predictor architecture, losses, training splits, and preprocessing.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The paper reports wall-clock measurements, multiple model sizes, several scheduling settings, ablations, and per-step diagnostic traces, but it does not include formal error bars or statistical significance tests. This choice is due to the high cost of repeated end-to-end RL runs; the reported diagnostics are provided to make the observed timing effects transparent.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 4 states that training-related experiments use a single node with 8 GPUs and 32GB memory per GPU, and the main tables report wall-clock rollout times. Appendix D reports preprocessing, predictor-training, and labeling overheads.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work is system-level research on rollout efficiency, uses public research assets, and does not involve human-subject experiments, private data collection, or release of a new high-risk model. Responsible release considerations are discussed in Appendix I.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Appendix I discusses positive impacts from reducing compute cost and energy use, as well as the possible negative impact that cheaper rollout generation can lower the cost of training more capable models. It also describes responsible-release choices that preserve the restrictions of underlying models and datasets.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release a new pretrained language model, image generator, scraped dataset, or user-facing generation service. The released artifact is research code and scripts, and users are instructed to obtain underlying models and datasets from their original sources under their existing terms.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper cites the creators of the public models, datasets, frameworks, and baselines used in the experiments, including Qwen3, Llama-3.1-8B-Instruct, `verl`, `MATH`, `DeepScaleR`, `GSM8K`, `MBPP`, `UltraFeedback`, and `GPQA`. Appendix I lists the main asset licenses and states that asset versions, licenses, and download/preprocessing instructions are documented in the released artifact.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The new asset is the LADDERS research code and reproducibility artifact, not a new dataset or model checkpoint. Appendix I describes the artifact contents, and the accompanying release includes configuration files, reproduction commands, and documentation.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The work does not involve crowdsourcing, new annotation labor, user studies, or other human-subject experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not conduct human-subject research, collect participant data, or involve crowdsourcing, so IRB approval or equivalent review is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLMs are central to the research problem and are explicitly described throughout the paper: LADDERS accelerates policy-model rollout generation, and Section 4 specifies the evaluated Qwen3 and Llama model families. No additional non-standard LLM usage beyond the core experimental methodology is required for the claims.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.