PRORSA: UNLEASHING THE POTENTIAL OF PREFILL-DECODE DISAGGREGATION IN AGENTIC REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Agentic reinforcement learning (RL) is commonly used to enhance the capabilities of large language models (LLMs) in solving complex multi-turn tasks. During rollout, agentic RL exhibits a workload pattern distinct from that of traditional RL: multi-turn interactions repeatedly introduce prefill requests, causing prefill and decode workloads to coexist throughout rollout. This workload makes prefill-decode (PD) disaggregation, a design widely used for online LLM serving, promising for speeding up agentic RL rollout. This paper shows that PD disaggregation can improve rollout efficiency by (a) mitigating interference among data-parallel attention modules, (b) enabling stage-specific parallelism, and (c) reducing end-to-end trajectory completion time. However, a fixed prefill-decode allocation may fail to sustain these benefits as workload composition varies across tasks and over time. We therefore present PRORSA, an agentic RL framework for prediction-guided prefill-decode resource allocation. PRORSA observes the lifecycle of every trajectory to forecast short-horizon prefill and decode demand. These forecasts then guide worker allocation decisions and trigger role switching of rollout engines, aligning prefill and decode capacity with anticipated workload demand. Across synchronous and asynchronous agentic RL settings, PRORSA achieves end-to-end rollout speedups of 1.11×–4.05× over prefill-decode colocation.

1 INTRODUCTION

Reinforcement learning (RL) has become a core stage of post-training for large language models (LLMs) (The Microsoft AI Team, 2026; DeepSeek-AI, 2025a; Qwen Team, 2025; GLM-5-Team, 2026; DeepSeek-AI, 2026; Kimi Team, 2026). Beyond single-turn reasoning, RL is increasingly applied to *agentic* scenarios—coding agents (Pan et al., 2024; Wei et al., 2025), computer use (Wang et al., 2025b; Huang et al., 2026; Lu et al., 2026), and deep research (Jin et al., 2025; Zheng et al., 2025)—where the model accomplishes a long-horizon task through multiple turns of interaction with the environment.

During RL training, the rollout phase generates samples from prompts (Sheng et al., 2025; Hu et al., 2024b). For agentic RL, a sample is no longer a single model response but a long-lived trajectory that interleaves LLM generation with tool execution across many turns. In this case, rollout can account for up to 70% of end-to-end iteration time, making its efficiency an important optimization target (Zhong et al., 2024b; Fu et al., 2025; Gao et al., 2025; Zhang et al., 2026; Chen et al., 2026b).

This paper studies how to improve agentic rollout efficiency through prefill-decode (PD) disaggregation, which places the two inference stages on separate GPU groups. The benefits of PD disaggregation have been well studied in online LLM serving, where it enables independent resource allocation and reduces interference between the two stages (Zhong et al., 2024a; Patel et al., 2024; Qin et al., 2025b; Hu et al., 2024a; Hong et al., 2025; Feng et al., 2025; Zhu et al., 2025). Its benefits in agentic RL rollout, however, have not been clearly characterized by prior works (GLM-5-Team, 2026; Chen et al., 2026a; Gao et al., 2026b).

Our analysis shows that agentic RL exhibits distinct characteristics that make PD disaggregation particularly beneficial. Specifically, agentic RL generates a continuous stream of prefill requests during rollout. The prefill workloads come from two sources: (1) *multi-turn interactions*, where

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

each trajectory repeatedly issues inference requests across interaction turns, creating prefill demand as new environment feedback is appended to its context, and (2) *asynchronous rollout*, where new samples are continuously injected into rollout engines rather than all at once to mitigate the long-tail problem (Fu et al., 2025; Zhong et al., 2025; Han et al., 2025; Zhou et al., 2025). As a result, prefill and decode requests coexist throughout the rollout phase (§3.1).

For these interleaved rollout workloads, we find three key benefits of PD disaggregation: (1) reducing GPU idleness caused by prefill-decode interference across data-parallel attention modules, (2) enabling independent parallelism strategies for prefill and decode to maximize throughput, and (3) reducing decode latency for long-tail trajectories.

However, applying PD disaggregation to agentic RL is not straightforward. Prefill and decode workloads vary across RL tasks, complicating the choice of a suitable prefill-to-decode instance ratio (PD ratio). Moreover, the prefill and decode demands change over time even within the same task, so a fixed PD ratio may fail to sustain a throughput advantage over colocation throughout training.

To address this challenge, we propose PRORSA, an agentic RL framework that combines PD disaggregation with adaptive worker allocation to improve rollout efficiency. The key insight of PRORSA is that the RL framework's visibility into trajectory lifecycles provides advance signals of upcoming inference demand. By monitoring sample dispatch, sandbox startup, tool execution, and reward completion, PRORSA forecasts short-horizon prefill and decode demand. Based on these forecasts, PRORSA adjusts the PD ratio by switching rollout workers between prefill and decode roles. This adjustment mechanism keeps the instance allocation aligned with changing workload demands, improving GPU utilization.

Our main contributions are as follows:

- We characterize the benefits and deployment challenges of PD disaggregation in agentic RL rollout through an in-depth workload and performance analysis.
- We introduce a prefill-decode role switching strategy driven by short-horizon workload forecasts to adapt resource allocation to changing rollout demand.
- We design and implement PRORSA, an agentic RL framework that adopts PD disaggregation to improve rollout efficiency. Experimental results across various agentic RL scenarios demonstrate that PRORSA achieves rollout speedups ranging from 1.11× to 4.05× over colocation in both synchronous and asynchronous settings.

2 BACKGROUND

2.1 AGENTIC RL TRAINING

In RL training, each iteration includes a *rollout* phase that generates trajectories using inference engines (Kwon et al., 2023; Zheng et al., 2024), a *reward* phase that scores the trajectories with predefined rules or reward models, and a *training* phase that updates the model with an RL algorithm such as PPO (Schulman et al., 2017), GRPO (Shao et al., 2024), or DAPO (Yu et al., 2025). Training begins once enough trajectories are scored, and updated weights are synchronized to rollout engines.

In agentic RL, a sample is no longer a single response but a multi-turn trajectory: in each turn the model emits an action (e.g., a tool call), the environment executes it, and the feedback is appended to the context for the next turn. For on-policy training, the rollout is *synchronous*: training cannot begin until all trajectories in a batch are complete, so long-tail trajectories leave GPUs idle. To mitigate this effect, large-scale systems often adopt *asynchronous* rollout (The Microsoft AI Team, 2026; DeepSeek-AI, 2026; GLM-5-Team, 2026; Fu et al., 2025; Zhong et al., 2025; Han et al., 2025), which continuously injects new samples and allows off-policy updates using trajectories generated by earlier model versions. In this case, a *staleness* bound limits the number of model versions a trajectory can span, and trajectories that fail to finish within this bound are dropped.

2.2 PREFILL-DECODE DISAGGREGATION

LLM inference consists of two stages: the *prefill* stage processes all prompt tokens in parallel and produces the first output token, while the *decode* stage generates the remaining tokens autoregressively. Prefill-decode disaggregation (Zhong et al., 2024a) places the two stages on separate in-

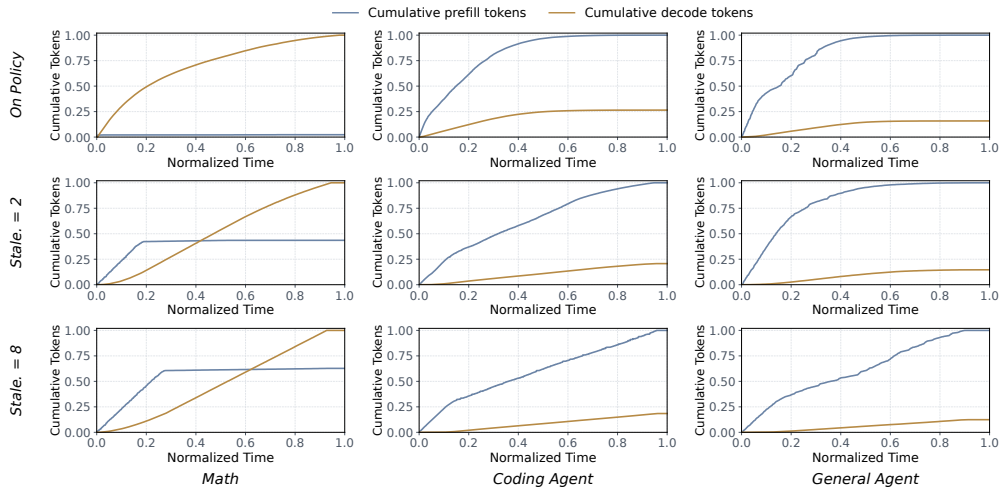


Figure 1: Normalized cumulative prefill and decode tokens for different RL tasks.

stances: prefill instances process prompts and transfer the produced *KV cache* to decode instances, which carry out generation. In online serving, disaggregation prevents decode from being blocked by prefill, and enables separate configuration of parallelism, accelerator types, and instance counts for prefill and decode to meet their respective latency SLOs (Zhong et al., 2024a; Patel et al., 2024).

3 Motivation and Challenges

In this section, we first characterize agentic RL workloads (§3.1), and then show the benefits of PD disaggregation (§3.2) under continuous prefill demand. Finally, we present the challenges of adopting PD disaggregation for dynamic RL rollout workloads (§3.3).

3.1 Workload Characterization of Agentic RL Rollout

Figure 1 shows the cumulative prefill and decode tokens within one rollout step for three representative tasks: math, coding agent, and general agent. The details of the tasks are described in §5.1. We vary the maximum staleness to capture the effects of asynchronous rollout. At the start of each step, trajectories carried over from the previous step must re-prefill their existing contexts with the latest model version (Fu et al., 2025; Xiaomi MiMo Team, 2026).

The tasks exhibit distinct workload patterns. For math, prefill is concentrated near the beginning of each step, primarily due to initial prompt processing and, in asynchronous settings, re-prefill of trajectories carried over from the previous step. In contrast, the two agentic tasks generate prefill requests throughout the step, and their prefill-token volumes substantially exceed their decode-token volumes. Under the on-policy setting, the token-processing rate drops sharply near the end of the step, revealing a long tail of straggling trajectories (Zhong et al., 2024b; 2025; Gao et al., 2025; Zhang et al., 2026; Zhou et al., 2025).

3.2 Motivating Experiments

Given this continuous prefill demand in agentic RL rollout, this paper shows that PD disaggregation can improve rollout efficiency. Using MiMo-V2.5-Pro (Xiaomi MiMo Team, 2026) and the experimental setup in §5.1, we identify three major sources of performance improvement:

Mitigating interference among data-parallel attention modules. Prior work has studied how PD colocation increases *decoding latency* through head-of-line blocking (Zhong et al., 2024a; Patel et al., 2024). However, we show that PD colocation can also significantly degrade *throughput* for mixture-of-experts (MoE) models. In MoE models, experts account for most model parameters, so their deployment commonly combines large-scale expert parallelism with multiple data-parallel (DP) attention modules (DeepSeek-AI, 2025b; Xiaomi MiMo Team, 2026; GLM-5-Team, 2026).

Table 1: Normalized per-server prefill and decode throughput for MiMo-V2.5-Pro. Decode uses a batch size of 16 per attention DP rank under all configurations.

Stage	Length	EP16	EP32	EP64
Prefill	8K	**3.24**	2.78	2.22
	64K	**2.77**	2.33	1.95
	256K	**1.76**	1.36	1.37
	512K	**1.29**	1.02	1.00
Decode	8K	1.20	**1.31**	1.29
	64K	1.17	**1.26**	1.25
	256K	1.10	**1.14**	1.13
	512K	**1.02**	1.01	1.00

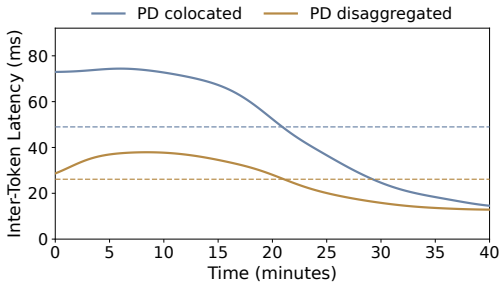
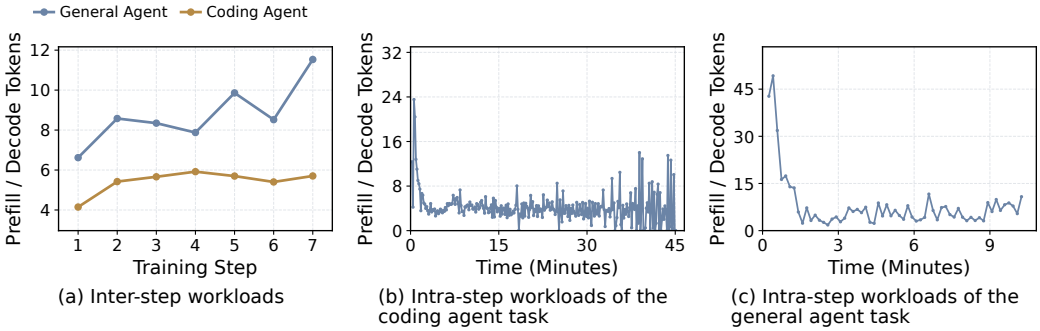


Figure 2: Inter-token latency of PD disaggregation versus colocation for synchronous rollout on the coding agent task. Dashed lines indicate the corresponding mean latencies.



(a) Inter-step workloads (b) Intra-step workloads of the coding agent task (c) Intra-step workloads of the general agent task

Figure 3: Prefill-to-decode token ratios across and within steps with a maximum staleness of 8.

In this case, when prefill and decode are colocated, their differing execution times cause interference across attention DP ranks: if one rank processes prefill while others process decode, the decoding ranks must wait before entering the synchronized MoE phase, leaving GPUs idle. Chunked prefill (Agrawal et al., 2024) shortens each stall but cannot eliminate the duration mismatch. This synchronization-induced idleness can reduce rollout throughput: for MiMo-V2.5-Pro on the two agentic tasks, we observe a GPU idle ratio of up to 26% due to prefill-decode interference (§5.3).

Independent parallelism strategies for prefill and decode. PD disaggregation also allows each stage to use its own parallelism strategy (Zhong et al., 2024a; Forys et al., 2026). Table 1 shows the normalized per-server throughput of prefill and decode under different parallelism strategies and input lengths using the testbed in §5.1. The results show that prefill consistently favors EP16, whereas decode favors EP32 for request lengths up to 256K and EP16 at 512K. Although EP64 does not maximize decode throughput, it provides more KV-cache capacity, which can be necessary for extremely long requests. These measurements motivate selecting parallelism separately for prefill and decode instances.

Reduced decode latency. RL rollout time is often dominated by a small number of long-tail trajectories, and prior work primarily addresses them by reducing decode latency (Zhong et al., 2024b; 2025; Gao et al., 2025; Zhang et al., 2026). PD disaggregation reduces decode latency by isolating decode from prefill interference. For synchronous rollout on the coding agent task, Figure 2 shows that PD disaggregation achieves significantly lower inter-token latency than colocation. The reduced decoding latency can shorten long-running trajectories, thus alleviating the long-tail problem.

3.3 Challenges

The benefits of PD disaggregation depend on assigning the right number of workers to each stage. However, as shown in Figure 1, workload composition varies substantially across tasks, requiring task-specific PD allocations. Moreover, this composition also varies during training. Figure 3(a) shows the prefill-to-decode token ratios across training steps with a maximum staleness of 8. The maximum-to-minimum ratio can reach 1.74×, as asynchronous sample dispatch leads to additional

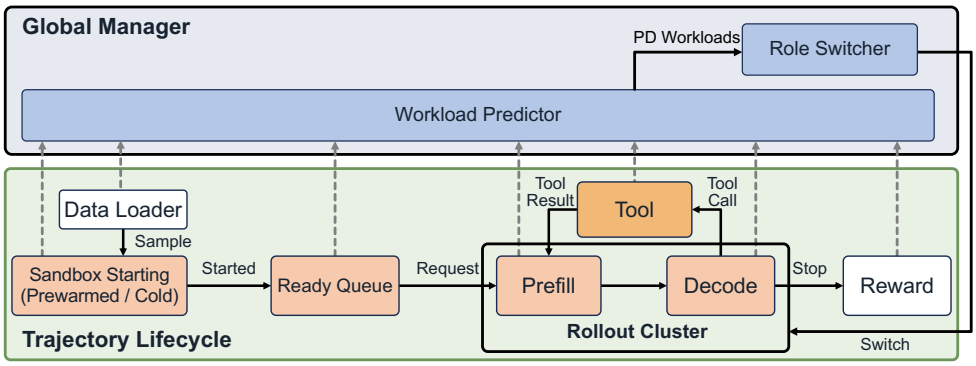


Figure 4: PRORSA architecture overview.

variability in rollout demand. Figure 3(b) and Figure 3(c) further show sharp workload fluctuations within the first training step at 10-second granularity. The complete workload analysis across different staleness settings is provided in Appendix A. Together, these variations motivate dynamically adjusting the PD worker ratio rather than maintaining a fixed allocation throughout training.

4 PRORSA DESIGN

To adapt prefill-decode disaggregation to dynamic rollout workloads, we propose PRORSA, an agentic RL framework with prediction-guided worker allocation. Figure 4 presents the architecture of PRORSA. The framework leverages its visibility into each trajectory's lifecycle, including sample dispatch, sandbox startup, inference requests, tool completion, and reward computation. Using these lifecycle signals, PRORSA forecasts short-horizon prefill demand and decode KV-cache requirements (§4.1) and dynamically switches rollout workers between the two roles (§4.2) to accommodate upcoming requests.

4.1 WORKLOAD PREDICTOR

4.1.1 TRAJECTORY STATE MACHINE

To forecast future demands, PRORSA traces each trajectory from sampling to final reward. Figure 4 shows the state machine of every trajectory. Each sampled trajectory starts a sandbox, either from the prewarm pool or through a cold startup. Once the sandbox is ready, the trajectory's first model request joins the ready queue. Next, admission to PREFILL creates a prefill arrival, and completion of prefill creates a decode admission and moves the trajectory to DECODE.

The decoded response either terminates the trajectory or contains a tool call. In the latter case, the trajectory enters TOOL and waits for the external tool or environment to return. The tool result is appended to the trajectory context, creating a new prefill request. If the decoded response reaches a stopping condition, the trajectory enters REWARD and contributes no further requests.

4.1.2 WORKLOAD PREDICTION

With the trajectory state machine, PRORSA can predict the prefill workload and decode-admission token volume from four sources: the data sampler, sandbox startup, the ready queue, and tool execution. The following summarizes each component and how it combines with in-flight work. All calculations incur negligible overhead. Due to space constraints, we leave the problem formulation and detailed equations to Appendix B.

- **Data sampler.** The sampler estimates how many trajectories currently in the reward phase will complete within the horizon. It applies an estimated training-admission rate (the fraction of reward-completed trajectories meeting training-admission criteria) and the sample-injection policy to infer new launches, then converts them into prefill demand using estimated prompt lengths. Newly sampled requests are assumed to hit the prewarm pool.

- **Sandbox startup.** For sandboxes already starting, the predictor distinguishes prewarm hits from cold starts. Prewarm hits are assumed to issue a request within the horizon, while historical cold-start durations determine each cold sandbox's expected contribution to the prefill workload.
- **Ready queue.** The predictor estimates prefill completion times from request arrival times, per-instance queue order, and profiled execution times under the selected configuration. Requests expected to complete prefill within the horizon contribute their prompt-token counts.
- **Tool execution.** For trajectories currently executing tools, the predictor uses historical tool durations and elapsed execution time to estimate when the next model request will arrive. Expected arrivals within the horizon contribute the estimated prompt length after the tool result is appended.
- **In-flight prefill requests.** Requests that are already in PREFILL and expected to complete prefill within the horizon contribute their prompt-token counts.
- **Decode admissions from prefill completion.** Requests expected to complete prefill within the horizon contribute their prompt lengths to the decode-admission token forecast. These token counts are converted to KV-cache requirements under the selected decode configuration, without predicting output lengths.
- **Aggregation and online correction.** The predictor sums the prefill contributions and refreshes the forecasts every S seconds. Observed arrivals and completions update duration statistics, training-admission outcomes update the acceptance-rate estimate, and forecast residuals drive an online bias correction.

4.2 PREFILL-DECODE ROLE SWITCHING

We select the parallelism configuration with the highest measured throughput separately for prefill and decode. For workloads with particularly long requests, a larger decode EP size is used to provide more KV-cache capacity. In agentic RL, KV-cache capacity limits concurrency for long requests, making it a key constraint on decode throughput. Therefore, the role switcher uses decode KV-cache requirements combined with prefill demand to set the PD instance ratio.

At time t, let $\mathcal{W}_t^{\mathrm{P}}$ denote the estimated number of prompt tokens to prefill over a prediction horizon of duration S. K_t^{D} is the current decode KV-cache footprint, and $\hat{K}_{t,S}^{\mathrm{D}}$ is the KV-cache space required by requests predicted to enter decode within the horizon. Over a recent sliding window, the switcher measures prefill throughput θ_t^{P} (prompt tokens per second per instance). Let $C_{\mathrm{KV}}^{\mathrm{D}}$ denote the usable KV-cache capacity per decode instance. The switcher computes the prefill instance count from throughput and the decode instance count solely from KV-cache capacity:

$$n_t^{\mathrm{P}} = \left\lceil \frac{\mathcal{W}_t^{\mathrm{P}}}{S \times \theta_t^{\mathrm{P}}}(1+\delta) \right\rceil, \qquad n_t^{\mathrm{D}} = \left\lceil \frac{K_t^{\mathrm{D}} + \hat{K}_{t,S}^{\mathrm{D}}}{C_{\mathrm{KV}}^{\mathrm{D}}}(1+\delta) \right\rceil, \tag{1}$$

where $\delta \geq 0$ is a shared safety margin. Within our short prediction horizon, we do not explicitly model KV-cache growth during decoding or cache reclamation upon request completion, as these are hard to predict. The safety margin δ provides KV-cache headroom to sustain decode throughput.

Let m^{P} and m^{D} be the numbers of GPUs per instance, and let G be the total GPU budget. Our objective is to maximize δ subject to the GPU budget constraint $m^{\mathrm{P}} n_t^{\mathrm{P}} + m^{\mathrm{D}} n_t^{\mathrm{D}} \leq G$. The resulting instance counts define the target PD ratio $n_t^{\mathrm{P}} : n_t^{\mathrm{D}}$. If the predicted demands cannot be met within the budget even at $\delta = 0$, the switcher instead chooses a budget-feasible integer allocation that minimizes the larger of the prefill throughput and decode KV-cache demand-to-capacity ratios. Appendix C details the switch execution procedure.

5 EVALUATION

We implement PRORSA on top of Verl (Sheng et al., 2025) and SGLang (Zheng et al., 2024). Our modifications comprise around 6.4K lines of Python code and support scheduling and role switching. In this section, we evaluate PRORSA in terms of overall performance (§5.2), sources of performance gains (§5.3), role-switching effectiveness (§5.4), and workload prediction accuracy (§5.5).

5.1 EXPERIMENTAL SETUP

Testbed. Experiments are conducted on a GPU cluster with 16 servers, each with 8 GPUs connected via NVLink. Each server is equipped with eight 400 Gbps RDMA NICs for inter-server connectivity.

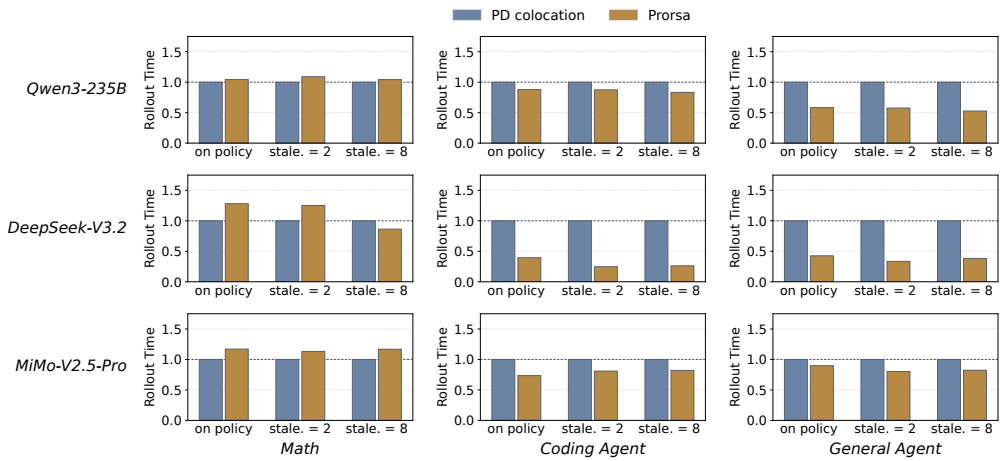


Figure 5: Overall performance of PRORSA and PD colocation.

Models. We use three models: (a) Qwen3-235B-A22B (Qwen Team, 2025), (b) DeepSeek-V3.2 (671B) (DeepSeek-AI, 2025c), and (c) MiMo-V2.5-Pro (1T) (Xiaomi MiMo Team, 2026). We enable chunked prefill (Agrawal et al., 2023) and use SGLang's recommended chunk sizes (4096 for Qwen3-235B and MiMo-V2.5-Pro, and 512 for DeepSeek-V3.2). All models use FP8 quantization. For each model, we select an optimal parallelism configuration that allows the model to fit within GPU memory while reserving sufficient KV-cache space for agentic RL rollout. We set the expert-parallel size to 16 for DeepSeek-V3.2 and MiMo-V2.5-Pro, and to 4 for Qwen3-235B. For attention tensor parallelism, Qwen3-235B uses TP=2 and MiMo-V2.5-Pro uses TP=8. For DeepSeek-V3.2, we disable attention tensor parallelism since it would duplicate the KV cache across TP ranks.

Workloads. We use the following three representative RL tasks.

- *Math*: single-turn mathematical problem solving tasks from AIME25 (Zhang & Math-AI, 2025).
- *Coding Agent*: software engineering tasks from SWE-smith (Yang et al., 2025).
- *General Agent*: general-purpose tasks from APEX-Agents (Vidgen et al., 2026), Toolathlon (Li et al., 2026b), and MCP-Atlas (Bandi et al., 2026).

All tasks use the GRPO algorithm (Shao et al., 2024), with a group size of 8 for the math task and 32 for the two agentic tasks. We use a 10-second workload prediction window in all experiments.

5.2 Overall Performance

Figure 5 compares the average rollout times of PRORSA and PD colocation across different settings. The results indicate that PRORSA reduces end-to-end rollout time on the evaluated agentic tasks. For the two agentic RL tasks, PRORSA delivers a 1.11×–4.05× speedup across different staleness settings, demonstrating the potential of PD disaggregation. Specifically, for DeepSeek-V3.2, which has 16 individual attention modules per engine, PRORSA improves rollout efficiency by 2.54×–4.05× on the coding agent task and 2.35×–2.99× on the general agent task. We attribute most of this speedup to reduced interference among attention modules, as detailed in §5.3. In contrast, on the math task, normalized rollout step times range from 0.86 to 1.28 relative to colocation, indicating a slowdown in most evaluated settings. This is due to much less overlap between prefill and decode: rollout first prefills all sequences and then executes successive decoding steps. For these workloads, PD disaggregation incurs KV-cache transfer and role-switching overheads that degrade performance.

5.3 Performance Breakdown

Improved GPU utilization. Figure 6 reports an average GPU idle ratio of 3.76%–59.98% under PD colocation due to the prefill-decode interference across attention DP ranks. For DeepSeek-V3.2 (DP=16), this interference is the dominant throughput bottleneck: a single prefill request can stall all other decode batches, yielding idle ratios of 36.21%–59.98%. This idleness is the primary contributor to rollout slowdown. Note that the reported idle ratios exclude GPU idleness arising

Table 2: Average end-to-end trajectory completion time (seconds) on the general agent task.

Model	Qwen3-235B			DeepSeek-V3.2			MiMo-V2.5-Pro		
Staleness	On-policy	2	8	On-policy	2	8	On-policy	2	8
PD colocation	133.88	169.14	176.55	713.05	723.96	939.54	253.17	198.08	252.36
PRORSA	78.98 (1.7× ↓)	86.96 (1.9× ↓)	80.40 (2.2× ↓)	119.47 (6.0× ↓)	123.21 (5.9× ↓)	128.12 (7.3× ↓)	195.35 (1.3× ↓)	158.90 (1.2× ↓)	216.42 (1.2× ↓)

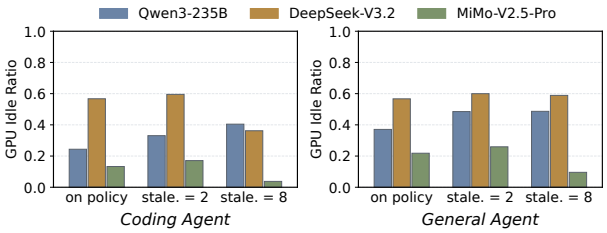


Figure 6: Average GPU idle ratio due to prefill-decode interference under PD colocation.

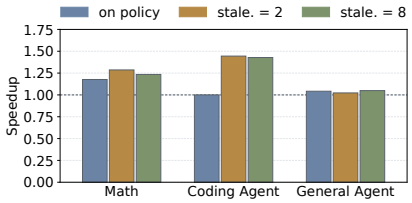


Figure 7: Effects of prediction-based role switching for MiMo-V2.5-Pro.

from scheduling overhead and cross-DP prefill imbalance, whose contributions are hard to quantify accurately. Therefore, these ratios cannot be directly translated into an end-to-end slowdown factor.

Stage-specific parallelism. PD disaggregation enables independent configuration of prefill and decode parallelism to maximize throughput. For the coding agent task, we double the expert-parallel size of decode instances for MiMo-V2.5-Pro and Qwen3-235B, yielding speedups of 1.04×–1.09×.

Reduced decode latency. Table 2 summarizes average trajectory completion times on the general agent task across different settings. Compared with PD colocation, PRORSA reduces end-to-end trajectory latency by 1.17×–7.33×. This improvement arises because PD disaggregation mitigates both types of prefill-decode interference: (a) cross-DP imbalance that blocks decode batches, as discussed earlier, and (b) head-of-line blocking caused by higher-priority prefill requests.

5.4 Ablation Study

Figure 7 evaluates the benefits of role switching by comparing PRORSA with static PD disaggregation for MiMo-V2.5-Pro. The static baseline uses the optimal prefill-to-decode instance ratios of 4:4, 5:3, and 3:5 for the math, coding agent, and general agent tasks, respectively. The results show that role switching yields speedups of 1.0×–1.44× across different settings. On the math task, an initial prefill phase for all samples is followed by prolonged decoding. A static PD ratio therefore underutilizes prefill and decode resources at different points in rollout.

The benefits of role switching depend on workload characteristics. Coding agent trajectories have an average end-to-end completion time of 846 seconds, 3.92× that of general agent trajectories (216 seconds). Longer trajectory lifetimes increase request concurrency, and asynchronous rollout further amplifies this effect through sample over-dispatch, placing greater pressure on decode KV-cache capacity and limiting throughput. Role switching alleviates this capacity bottleneck by increasing the number of decode instances when needed. In contrast, the general agent task benefits less from role switching due to lower KV-cache pressure.

5.5 Prediction Accuracy

Figure 8 reports prediction accuracy for MiMo-V2.5-Pro on the coding and general agent tasks across different staleness settings. We evaluate prefill and decode-admission forecasts separately using weighted absolute percentage error (WAPE), defined as the sum of absolute prediction errors divided by the sum of measured token counts across all windows. Across all settings, WAPE ranges from 5.40% to 14.37% for prefill and from 5.19% to 13.69% for decode. This level of accuracy is sufficient to guide role switching. The higher prediction errors on the general agent task reflect greater variability in tool-execution times.

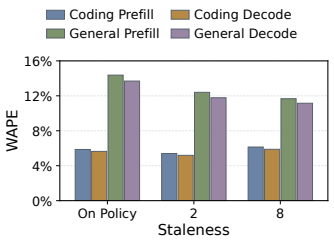


Figure 8: Prediction WAPE for MiMo-V2.5-Pro across staleness settings.

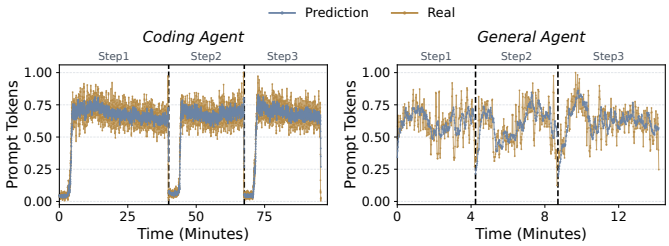


Figure 9: Predicted and measured normalized prefill workloads for MiMo-V2.5-Pro on the coding and general agent tasks. Dashed lines mark training-step boundaries.

Figure 9 further compares the prefill workload forecasts of PRORSA with the measured workloads over the first three training steps with staleness set to 8. The results show that, by monitoring all trajectories, PRORSA produces forecasts that align with the measured workloads.

6 Related Work

RL frameworks. RL frameworks improve performance through resource and trajectory scheduling (Sheng et al., 2025; Hu et al., 2024b; ROLL Team, 2025; Wang et al., 2025d;c; Hu et al., 2026; Gao et al., 2025; Zhong et al., 2024b; Mei et al., 2024; Wu et al., 2026), asynchronous rollout (Fu et al., 2025; Zhong et al., 2025; Han et al., 2025; Zhou et al., 2025), and speculative decoding (He et al., 2025; Qin et al., 2025a; Chen et al., 2025; Xu et al., 2026a). Systems also adapt rollout parallelism (Gao et al., 2025; Tan et al., 2026; Zhang et al., 2026) or multiplex prefill and decode in place (Wang et al., 2026a), and a few production RL systems already adopt PD disaggregation for agentic RL rollout (GLM-5-Team, 2026; Chen et al., 2026a; Gao et al., 2026b) without in-depth analysis. Our work characterizes the benefits and deployment challenges of PD disaggregation for agentic RL, and introduces role switching to adapt the PD ratio to changing rollout demand.

Prefill-decode disaggregation. PD disaggregation separates prefill and decode to avoid interference and enable stage-specific configuration (Zhong et al., 2024a; Patel et al., 2024; Qin et al., 2025b; Lin et al., 2024; Hong et al., 2025; Feng et al., 2025; Zhu et al., 2025; Forys et al., 2026; Wang et al., 2026b). Variants include hybrid aggregation and disaggregation (Wang et al., 2025a), intra-GPU multiplexing (Cui et al., 2025; Gao et al., 2026a; Xu et al., 2026b), refined splits for multi-turn prefills (Li et al., 2026c), adaptive multi-round prefill scheduling (He et al., 2026), and load-driven PD autoscaling (Liao et al., 2026). In contrast to these serving systems, PRORSA applies PD disaggregation to agentic RL rollout and uses workload predictions based on RL lifecycle signals.

LLM workload prediction. Existing LLM serving systems predict workloads for provisioning, scheduling, and model prewarming (Jiang et al., 2025; Fan et al., 2026; Jiang et al., 2026; Jaiswal et al., 2025; Lai et al., 2025; Lou et al., 2026; Tiwari et al., 2026; Li et al., 2026a). RL frameworks also exploit workload estimates: OrchestrRL (Tan et al., 2026) forecasts remaining rollout workloads, while Seer (Qin et al., 2025a), RhymeRL (He et al., 2025), and RollPacker (Gao et al., 2025) use online probes or historical workload statistics to guide scheduling and resource allocation. For agentic RL, Heddle (Zhang et al., 2026) predicts trajectory completion times, while Libra (Chen et al., 2026b) uses tool-return signals to estimate remaining work and guide scheduling. In contrast, PRORSA combines signals from reward-driven sample injection, sandbox startup, and tool returns to forecast short-horizon, turn-level prefill and decode demand in agentic RL rollout.

7 Conclusion

Agentic RL introduces continuous prefill requests, making PD disaggregation beneficial for the rollout phase. However, our experiments show that a static prefill-decode ratio leads to resource underutilization due to dynamic rollout workloads. PRORSA employs an adaptive PD role-switching strategy to accommodate fluctuating demand, guided by workload forecasts derived from trajectory lifecycle signals. Our evaluation demonstrates that PRORSA effectively unlocks the performance potential of PD disaggregation during rollout, achieving up to 4.05× the throughput of PD colocation.

8 AI USE STATEMENT

In this work, we used generative AI tools to help develop theoretical models or conceptual frameworks, implement methods, assist with translation, support qualitative and thematic data analysis, and interpret results. We did not use generative AI tools to generate synthetic datasets, formulate mathematical claims, propose or refine hypotheses, design or provide feedback on research methodology or experiments, or clean and reformat datasets. The tasks of providing critical ingredients for mathematical proofs and assisting with proof writing are not applicable to this work. Additionally, we used generative AI tools to create or modify scientific figures or images, create or edit software code, draft parts of the paper, summarize or analyze existing literature, brainstorm ideas, search for information, and edit the paper to improve readability. We have reviewed all AI-assisted work. Specifically, we checked AI-assisted conceptual formulations and method descriptions for consistency with the system design and implementation. We reviewed and tested AI-assisted code for correctness, and cross-checked figures, numerical summaries, and interpretations against the underlying experimental records. We also verified citations and descriptions of prior work against the original sources. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, and Ramachandran Ramjee. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. *arXiv preprint arXiv:2308.16369*, 2023.

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming Throughput-Latency tradeoff in LLM inference with Sarathi-Serve. In *USENIX OSDI*, 2024.

Chaithanya Bandi, Razvan-Gabriel Dumitru, Ben Hertzberg, Divyansh Agarwal, Geobio Boo, Tejas Polakam, Sami Hassaan, Jeff Da, HiJae Kim, Vipul Gupta, Manasi Sharma, Andrew Park, Martin Dimakis, Ernesto Gabriel Hernandez Montoya, Dan Rambado, Ivan Salazar, Rafael Cruz, MohammadHossein Rezaei, Chetan Rane, Ben Levin, Daniel Yue Zhang, Brad Kenstler, and Bing Liu. Mcp-atlas: A large-scale benchmark for tool-use competency with real mcp servers. *arXiv preprint arXiv:2602.00933*, 2026.

Aili Chen, Aonian Li, Baichuan Zhou, Bangwei Gong, Binyang Jiang, Boji Dan, Changhao Zhang, Changqing Yu, Chao Wang, Cheng Ma, Cheng Zhong, Cheng Zhu, Chengjun Xiao, Chengyi Yang, Chengyu Du, Chenyang Zhang, Chi Zhang, Chuangyi Huang, Chunhao Zhang, Chunhui Du, Chunyu Zhao, Congchao Guo, Da Chen, Deming Ding, Dianjun Sun, Dong Li, Dongyu Zhang, Enhui Yang, Fei Yu, Guang Zheng, Guodong Zheng, Guohong Li, Haichao Zhu, Haigang Zhou, Haimo Zhang, Han Ding, Hao Zhang, Haohai Sun, Haolin Lyu, Haonan Lu, Haoyu Wang, Huajie Shi, Huiyang Li, Jiacheng Chen, Jian Zhang, Jiaqi Zhuang, Jiaren Cai, Jiaxin Pan, Jiayao Li, Jiayuan Song, Jichuan Zhang, Jie Wang, Jihao Gu, Jin Zhu, Jingwei Dong, Jingyang Li, Jingyu Zhang, Jingze Zhuang, Jinhao Tian, Jinli Liu, Jinyi Hu, Jun Tao, Jun Zhang, Junbin Ruan, Junhao Xu, Junjie Yan, Junteng Liu, Junxian He, Kang Xu, Ke Ji, Ke Yang, Kecheng Xiao, Keyu Duan, Keyu Li, Le Han, Letian Ruan, Li Yuan, Lianfei Yu, Liheng Feng, Lijie Mo, Lin Li, Linge Du, Lingye Bao, Lingyu Yang, Lingyuan Zhou, Loki, Lu Chen, Lunbin Zeng, Ming Li, Ming Zhong, Mingliang Tao, Mingyuan Chi, Mujie Lin, Nan Hu, Ningxin Chen, Peiyin Zhu, Peng Gao, Pengcheng Gao, Pengfei Li, Penglin Li, Pengyu Zhao, Qibin Ren, Qibing Ren, Qidi Xu, Qihan Ren, Qile Li, Qin Wang, Quanliang Chen, Qunhong Zeng, Rong Tian, Rongxin Guo, Rui Dong, Ruitao Leng, Ruize Zhang, Shanqi Liu, Shaoxiang Chen, Shaoyu Chen, Sheng Jia, Shun Yao, Shuoran Zhao, Shuqi Yu, Sichen Li, Sicheng Pan, Songquan Zhu, Tengfei Li, Tian Xie, Tiancheng Qin, Tianle Li, Tianrun Liang, Wei Liu, Weiqi Xu, Weitao Li, Weixiang Chen, Weiyu Cheng, Weiyu Zhang, Wenhu Chen, Wenqian Zhao, Xiancai Chen, Xiangjun Song, Xiangyuan Wang, Xianzhen Luo, Xiao Luo, Xiao Su, Xiaobo Li, Xiaodong Han, Xiaojie Wu, Xihao Song, Xingyi Han, Xinyu Guan, Xuan Lu, Xun Zou, Xunhao Lai, Xutong Li, Xuyang Shen, Yan Gong, Yan Ma, Yang Jiao, Yang Wang, Yang Xu, Yangsen Wang, Ye Tang, Yicheng Chen, Yihang Wang, Yinran Qiu, Yiqi Shi, Yiting Guo, Yiwen Huang, Yixuan Wang, Yongyi Hu, Yu Gao, Yu Zhang, Yuan Li, Yuanxiang Ying, Yuanzhen Zhang, Yubo Wang, Yuchen Song, Yufeng Yang,

Yuhang Meng, Yuhang Miao, Yuhao Li, Yujie Liu, Yulin Hu, Yunan Huang, Yunji Li, Yunyi Huang, Yusen Zhang, Yusu Hong, Yutao Xie, Yutong Zhang, Yuwen Liao, Yuxuan Shi, Yuze Wenren, Zebin Li, Zehan Li, Zejian Luo, Zeyu Jin, Zeyuan Sun, Zhanpeng Zhou, Zhaochen Su, Zhendong Li, Zhengmao Zhu, Zhengyuan Peng, Zhenhua Fan, Zhi Zhang, Zhichao Xu, Zhiheng Lv, Zhikang Xu, Zhitao He, Zhiwei He, Zhongyuan Li, Zibo Gao, Zijia Wu, Zijian Song, Zijian Zhou, Zijun Sun, Zishan Huang, Ziying Chen, and Ziyue Ge. The minimax-m2 series: Mini activations unleashing max real-world intelligence. *arXiv preprint arXiv:2605.26494*, 2026a.

Kaiwen Chen, Xin Tan, Jingzong Li, and Hong Xu. Libra: Efficient resource management for agentic rl post-training. *arXiv preprint arXiv:2606.03077*, 2026b.

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.

Weihao Cui, Yukang Chen, Han Zhao, Ziyi Xu, Quan Chen, Xusheng Chen, Yangjie Zhou, Shixuan Sun, and Minyi Guo. Optimizing slo-oriented llm serving with pd-multiplexing. *arXiv preprint arXiv:2504.14489*, 2025.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025a.

DeepSeek-AI. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2025b.

DeepSeek-AI. Deepseek-v3.2: Pushing the frontier of open large language models, 2025c.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Yunqian Fan, Shihao Bai, Ruihao Gong, Zaijun Wang, and Rui Fan. Pillm: Resource-efficient llm inference using workload prediction. In *EuroSys*, 2026.

Jingqi Feng, Yukai Huang, Rui Zhang, Sicheng Liang, Ming Yan, and Jie Wu. Windserve: Efficient phase-disaggregated llm serving with stream-based dynamic scheduling. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pp. 1283–1295, 2025.

Przemyslaw Forys, Haoran Wu, Can Xiao, Jiayi Nie, Tony Liu, Rika Antonova, Timothy Jones, Robert Mullins, Wayne Luk, Aaron Zhao, and George A. Constantinides. When does disaggregation pay? simulating prefill–decode–attention–ffn specialization for agentic llm inference. *arXiv preprint arXiv:2608.03741*, 2026.

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

Lei Gao, Chaoyi Jiang, Hossein Entezari Zarch, Daniel Wong, Mark Hill, and Murali Annavaram. Duetserve: Harmonizing prefill and decode for llm serving via adaptive gpu multiplexing. *arXiv preprint arXiv:2511.04791*, 2026a.

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.

Wei Gao, Yuheng Zhao, Tianyuan Wu, Shaopan Xiong, Weixun Wang, Dakai An, Lunxi Cao, Dilxat Muhtar, Zichen Liu, Haizhou Zhao, Ju Huang, Siran Yang, Yongbin Li, Wenbo Su, Jiamang Wang, Lin Qu, Bo Zheng, and Wei Wang. RollArt: Disaggregated Multi-Task agentic RL training at scale. In *USENIX OSDI*, 2026b.

GLM-5-Team. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Zhenyu Han, Ansheng You, Haibo Wang, Kui Luo, Guang Yang, Wenqi Shi, Menglong Chen, Sicheng Zhang, Zeshun Lan, Chunshi Deng, et al. Asyncflow: An asynchronous streaming rl framework for efficient llm post-training. *arXiv preprint arXiv:2507.01663*, 2025.

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.

Wenhao He, Youhe Jiang, Penghao Zhao, Quanqing Xu, Eiko Yoneki, Bin Cui, and Fangcheng Fu. Efficient multi-round llm inference over disaggregated serving. *arXiv preprint arXiv:2602.14516*, 2026.

Ke Hong, Lufang Chen, Zhong Wang, Xiuhong Li, Qiuli Mao, Jianping Ma, Chao Xiong, Guanyu Wu, Buhe Han, Guohao Dai, et al. semi-pd: Towards efficient llm serving via phase-wise disaggregated computation and unified storage. *arXiv preprint arXiv:2504.19867*, 2025.

Cunchen Hu, Heyang Huang, Liangliang Xu, Xusheng Chen, Jiang Xu, Shuang Chen, Hao Feng, Chenxi Wang, Sa Wang, Yungang Bao, et al. Inference without interference: Disaggregate llm inference for mixed downstream workloads. *arXiv preprint arXiv:2401.11181*, 2024a.

Jian Hu, Xibin Wu, Wei Shen, Jason Klein Liu, Zilin Zhu, Weixun Wang, Songlin Jiang, Haoran Wang, Hao Chen, Bin Chen, et al. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework. *arXiv preprint arXiv:2405.11143*, 2024b.

Tianhao Hu, Xiangcheng Liu, Yuchun Miao, Youshao Xiao, Hongyu Zang, Yang Zheng, Xuan Huang, Jinrui Ding, Yufei Zhang, Yu Yang, Yi-Kai Zhang, Yueqing Sun, Chengcheng Han, Xiandi Ma, Wei Wang, Qi Gu, Yerui Sun, Yuchen Xie, and Xunliang Cai. Dora: A scalable asynchronous reinforcement learning system for language model training. *arXiv preprint arXiv:2604.26256*, 2026.

Mianqiu Huang, Taofeng Xue, Chong Peng, Jinrui Ding, Jie Yang, Sicheng Fan, Jiale Hong, Yufei Gao, Xiaocheng Zhang, Linsen Guo, Xin Yang, Dengchang Zhao, Yuchen Xie, Peng Pei, Xunliang Xie, and Xipeng Qiu. Evocua-1.5: Online reinforcement learning for multi-turn computer-use agents. *arXiv preprint arXiv:2607.09773*, 2026.

Shashwat Jaiswal, Kunal Jain, Yogesh Simmhan, Anjaly Parayil, Ankur Mallick, Rujia Wang, Renee St. Amant, Chetan Bansal, Victor Ruhle, Anoop Kulkarni, Steve Kofsky, and Saravan Rajmohan. Sageserve: Optimizing llm serving on cloud data centers with forecast aware auto-scaling. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, December 2025.

Youhe Jiang, Fangcheng Fu, Taiyi Wang, Guoliang He, and Eiko Yoneki. Oserve: Accelerating llm serving via spatial-temporal workload orchestration. *arXiv preprint arXiv:2602.12151*, 2026.

Zhihan Jiang, Yujie Huang, Guangba Yu, Junjie Huang, Jiazhen Gu, and Michael R. Lyu. Hierarchical prediction-based management for lmaas systems. *arXiv preprint arXiv:2504.03702*, 2025.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Kimi Team. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *ACM SOSP*, pp. 611–626, 2023.

Ruiqi Lai, Hongrui Liu, Chengzhi Lu, Zonghao Liu, Siyu Cao, Siyang Shao, Yixin Zhang, Luo Mai, and Dmitrii Ustiugov. Tokenscale: Timely and accurate autoscaling for disaggregated llm serving with token velocity. *arXiv preprint arXiv:2512.03416*, 2025.

Hanchen Li, Runyuan He, Qiuyang Mang, Qizheng Zhang, Huanzhi Mao, Xiaokun Chen, Hangrui Zhou, Huanchen Zhang, Alvin Cheung, Joseph Gonzalez, and Ion Stoica. Continuum: Efficient and robust multi-turn llm agent scheduling with kv cache time-to-live. *arXiv preprint arXiv:2511.02230*, 2026a.

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, Junteng Liu, Zhaochen Su, Yiyang Guo, Fan Zhou, Lueyang Zhang, Juan Michelini, Xingyao Wang, Xiang Yue, Shuyan Zhou, Graham Neubig, and Junxian He. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. *arXiv preprint arXiv:2510.25726*, 2026b.

Zongze Li, Jingyu Liu, Zhen Xu, Yineng Zhang, Tahseen Rabbani, and Ce Zhang. Not all prefills are equal: Ppd disaggregation for multi-turn llm serving. *arXiv preprint arXiv:2603.13358*, 2026c.

Junhan Liao, Minxian Xu, Wanyi Zheng, Yan Wang, Kejiang Ye, Rajkumar Buyya, and Chengzhong Xu. Dopd: A dynamic pd-disaggregation architecture for maximizing goodput in llm inference serving. *arXiv preprint arXiv:2511.20982*, 2026.

Bin Lin, Chen Zhang, Tao Peng, Hanyu Zhao, Wencong Xiao, Minmin Sun, Anmin Liu, Zhipeng Zhang, Lanbo Li, Xiafei Qiu, Shen Li, Zhigang Ji, Tao Xie, Yong Li, and Wei Lin. Infinite-llm: Efficient llm service for long context with distattention and distributed kvcache. *arXiv preprint arXiv:2401.02669*, 2024.

Chiheng Lou, Sheng Qi, Rui Kang, Yong Zhang, Chen Sun, Pengcheng Wang, Xuanzhe Liu, and Xin Jin. Warmserve: Enabling one-for-many gpu prewarming for multi-llm serving. *arXiv preprint arXiv:2512.09472*, 2026.

Dunjie Lu, Shuai Bai, Tianyi Bai, Sicheng Fan, Chang Gao, Jian Guan, Feng Hu, Mianqiu Huang, Xingyang Huang, Yizhen Jiang, Yuheng Jing, Dehui Kong, Ning Li, Dayiheng Liu, Shixuan Liu, Zheng Liu, Que Shen, Bowen Wang, Junli Wang, Chencan Wu, Rui Xie, Tianbao Xie, Zhihui Xie, Haiyang Xu, An Yang, Tao Yu, Wenzhen Yuan, Xi Zhang, Zhenru Zhang, Mingkang Zhu, Zhaoqing Zhu, Yizhong Cao, Kai Dang, Binyuan Hui, Kaixin Li, Junyang Lin, Haiquan Wang, Zekun Wang, Yiheng Xu, Fan Yan, Mengqi Yuan, Danyang Zhang, Jiajun Zhang, Zhipeng Zhang, Fan Zhou, and Fan Zhou. Qwen-cua: Native computer use for (almost) everything. *arXiv preprint arXiv:2608.02352*, 2026.

Zhiyu Mei, Wei Fu, Kaiwei Li, Guangju Wang, Huanchen Zhang, and Yi Wu. Real: Efficient rlhf training of large language models with parameter reallocation. *arXiv preprint arXiv:2406.14088*, 2024.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym, 2024.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Inigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *isca*, June 2024.

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*, 2025a.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation — a KVCache-centric architecture for serving LLM chatbot. In *USENIX Conference on File and Storage Technologies*, pp. 155–170, 2025b.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

ROLL Team. Reinforcement Learning Optimization for Large-Scale Learning: An Efficient and User-Friendly Scaling Library. *arXiv preprint arXiv:2506.06122*, 2025.

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

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

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.

Xin Tan, Yicheng Feng, Yu Zhou, Yimin Jiang, Yibo Zhu, and Hong Xu. Orchestrrl: Dynamic compute and network orchestration for disaggregated rl. *arXiv preprint arXiv:2601.01209*, 2026.

The Microsoft AI Team. Mai-thinking-1: Building a hill-climbing machine. Technical report, Microsoft AI, 2026. URL https://microsoft.ai/pdf/mai-thinking-1.pdf.

Shubham Tiwari, Tapan Chugh, Nash Rickert, Simon Peter, Ratul Mahajan, and Haiying Shen. Cachewise: Understanding workloads and optimizing kvcache management for efficiently serving llm coding agents. *arXiv preprint arXiv:2606.16824*, 2026.

Bertie Vidgen, Austin Mann, Abby Fennelly, John Wright Stanly, Lucas Rothman, Marco Burstein, Julien Benchek, David Ostrofsky, Anirudh Ravichandran, Debnil Sur, Neel Venugopal, Alannah Hsia, Isaac Robinson, Calix Huang, Olivia Varones, Daniyal Khan, Michael Haines, Austin Bridges, Jesse Boyle, Koby Twist, Zach Richards, Chirag Mahapatra, Brendan Foody, and Osvald Nitski. Apex-agents. *arXiv preprint arXiv:2601.14242*, 2026.

Chao Wang, Pengfei Zuo, Zhangyu Chen, Yunkai Liang, Zhou Yu, and Ming-Chang Yang. Prefill-decode aggregation or disaggregation? unifying both for goodput-optimized llm serving. *arXiv preprint arXiv:2508.01989*, 2025a.

Guowei Wang, Chaokun Yang, Zhenxuan Pan, Yipeng Wei, Yuhong Guo, Minghua Zhu, Zhechuan Zhang, Shuo Wan, and Xiaowei Zhu. Ainfer-pd: Communication-safe in-place prefill-decode multiplexing for distributed moe rollouts. *arXiv preprint arXiv:2609.00993*, 2026a.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, Wanjun Zhong, Yining Ye, Yujia Qin, Yuwen Xiong, Yuxin Song, Zhiyong Wu, Aoyan Li, Bo Li, Chen Dun, Chong Liu, Daoguang Zan, Fuxing Leng, Hanbin Wang, Hao Yu, Haobin Chen, Hongyi Guo, Jing Su, Jingjia Huang, Kai Shen, Kaiyu Shi, Lin Yan, Peiyao Zhao, Pengfei Liu, Qinghao Ye, Renjie Zheng, Shulin Xin, Wayne Xin Zhao, Wen Heng, Wenhao Huang, Wenqian Wang, Xiaobo Qin, Yi Lin, Youbin Wu, Zehui Chen, Zihao Wang, Baoquan Zhong, Xinchun Zhang, Xujing Li, Yuanfan Li, Zhongkai Zhao, Chengquan Jiang, Faming Wu, Haotian Zhou, Jinlin Pang, Li Han, Qi Liu, Qianli Ma, Siyao Liu, Songhua Cai, Wenqi Fu, Xin Liu, Yaohui Wang, Zhi Zhang, Bo Zhou, Guoliang Li, Jiajun Shi, Jiale Yang, Jie Tang, Li Li, Qihua Han, Taoran Lu, Woyu Lin, Xiaokang Tong, Xinyao Li, Yichi Zhang, Yu Miao, Zhengxuan Jiang, Zili Li, Ziyuan Zhao, Chenxin Li, Dehua Ma, Feng Lin, Ge Zhang, Haihua Yang, Hangyu Guo, Hongda Zhu, Jiaheng Liu, Junda Du, Kai Cai, Kuanye Li, Lichen Yuan, Meilan Han, Minchao Wang, Shuyue Guo, Tianhao Cheng, Xiaobo Ma, Xiaojun Xiao, Xiaolong Huang, Xinjie Chen, Yidi Du, Yilin Chen, Yiwen Wang, Zhaojian Li, Zhenzhu Yang, Zhiyuan Zeng, Chaolin Jin, Chen Li, Hao Chen, Haoli Chen, Jian Chen, Qinghao Zhao, and Guang Shi. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*, 2025b.

Jinghui Wang, Shaojie Wang, Yinghan Cui, Xuxing Chen, Chao Wang, Xiaojiang Zhang, Minglei Zhang, Jiarong Zhang, Wenhao Zhuang, Yuchen Cao, et al. Seamlessflow: A trainer agent isolation rl framework achieving bubble-free pipelines via tag scheduling. *arXiv preprint arXiv:2508.11553*, 2025c.

Yida Wang, Xiuhong Li, Jianping Ma, Gan Sun, Yunshen Xu, Buhe Han, Jingxu Ng, Yuhao Luo, Ke Hong, Guohao Dai, Boxun Li, and Yu Wang. Pdd: Unleashing economical and flexible heterogeneous llm inference via cross-datacenter prefill-decode disaggregation. *arXiv preprint arXiv:2609.13161*, 2026b.

Zhixin Wang, Tianyi Zhou, Liming Liu, Ao Li, Jiarui Hu, Dian Yang, Yinhui Lu, Jinlong Hou, Siyuan Feng, Yuan Cheng, et al. Distflow: A fully distributed rl framework for scalable and efficient llm post-training. *arXiv preprint arXiv:2507.13833*, 2025d.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution. *arXiv preprint arXiv:2502.18449*, 2025.

Tianyuan Wu, Lunxi Cao, Yining Wei, Wei Gao, Yuheng Zhao, Dakai An, Shaopan Xiong, Zhiqiang Lv, Ju Huang, Siran Yang, Yinghao Yu, Jiamang Wang, Lin Qu, and Wei Wang. Weave: Efficient co-scheduling for disaggregated rl post-training. 2026.

Xiaomi MiMo Team. Mimo-v2.5-pro. https://huggingface.co/collections/XiaomiMiMo/mimo-v25, 2026.

Ryan Xu, Atlas Zhao, David Bao, and Frank Du. War: Workload-aware rollouts for synchronous agentic reinforcement learning. *arXiv preprint arXiv:2607.17299*, 2026a.

Yi Xu, Ehsan K. Ardestani, Wenyin Fu, Martin Schatz, Krishna Malladi, Zhan Shu, Adnan Aziz, Shobhit Kanaujia, Ajit Mathews, and Chunqiang Tang. Crossflow: Prefill-decode elasticity for agentic llm serving. *arXiv preprint arXiv:2609.27085*, 2026b.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents. *arXiv preprint arXiv:2504.21798*, 2025.

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.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2025, 2025.

Zili Zhang, Yinmin Zhong, Chengxu Yang, Chao Jin, Bingyang Wu, Xinming Wei, Yuliang Liu, and Xin Jin. Heddle: A distributed orchestration system for agentic rl rollout. *arXiv preprint arXiv:2603.28101*, 2026.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Neural Information Processing Systems*, 2024.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. *arXiv preprint arXiv:2504.03160*, 2025.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *USENIX OSDI*, 2024a.

Yinmin Zhong, Zili Zhang, Bingyang Wu, Shengyu Liu, Yukun Chen, Changyi Wan, Hanpeng Hu, Lei Xia, Ranchen Ming, Yibo Zhu, et al. Optimizing rlhf training for large language models with stage fusion. *arXiv preprint arXiv:2409.13221*, 2024b.

Yinmin Zhong, Zili Zhang, Xiaoniu Song, Hanpeng Hu, Chao Jin, Bingyang Wu, Nuo Chen, Yukun Chen, Yu Zhou, Changyi Wan, et al. Streamrl: Scalable, heterogeneous, and elastic rl for llms with disaggregated stream generation. *arXiv preprint arXiv:2504.15930*, 2025.

Yuzhen Zhou, Jiajun Li, Yusheng Su, Gowtham Ramesh, Zilin Zhu, Xiang Long, Chenyang Zhao, Jin Pan, Xiaodong Yu, Ze Wang, Kangrui Du, Jialian Wu, Ximeng Sun, Jiang Liu, Qiaolin Yu, Hao Chen, Zicheng Liu, and Emad Barsoum. April: Active partial rollouts in reinforcement learning to tame long-tail generation. *arXiv preprint arXiv:2509.18521*, 2025.

Kan Zhu, Yufei Gao, Yilong Zhao, Liangyu Zhao, Gefei Zuo, Yile Gu, Dedong Xie, Tian Tang, Qinyu Xu, Zihao Ye, Keisuke Kamahori, Chien-Yu Lin, Ziren Wang, Stephanie Wang, Arvind Krishnamurthy, and Baris Kasikci. Nanoflow: Towards optimal large language model serving throughput, 2025.

A Additional Motivation Experiments

We extend the workload analysis in §3.3 with the complete plots of inter-step and intra-step workload variation for the two agentic tasks under different configurations.



Figure 10: Inter-step prefill-to-decode token ratios for the general agent and coding agent tasks over seven training steps.

Inter-step variation. Figure 10 shows that workload variation occurs across all settings. Even in the relatively stable on-policy setting, the maximum ratio is 1.20× the minimum.

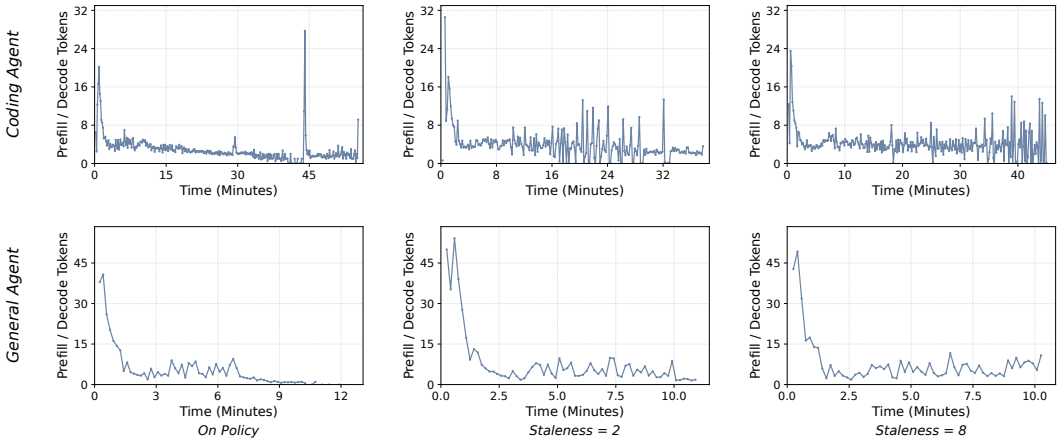


Figure 11: Intra-step prefill-to-decode token ratios within the first training step, measured over 10-second windows.

Intra-step variation. Figure 11 reveals a similar pattern of workload variation across all settings. Across all settings, the first rollout step always begins with a prefill-heavy phase, followed by a sharp decline in the token ratio.

B Workload Prediction

This section formalizes the workload predictor in §4.1 and details the procedure summarized in §4.1.2. The predictor estimates prefill demand and decode-admission token volume from the data sampler, sandbox startup, the ready queue, tool execution, and observed in-flight requests.

B.1 Problem Formulation

At time t, the predictor estimates two demand signals over the short horizon $[t, t+S)$ (default $S = 10\,\text{s}$): a prefill workload estimate for worker allocation and the prompt-token volume of requests expected to be newly admitted to decode. Throughout this section, x_j denotes the observed or predicted full prompt length of request j, including any cached prefix; we do not subtract cached or already processed tokens from this length. For simplicity, we do not model a trajectory entering the prefill and decode stages more than once within the prediction horizon.

Input. The input is an atomic snapshot of the rollout framework at time t, including the lifecycle record T_t of every live trajectory: its current state, context length, elapsed time in that state, and group membership under group-based algorithms. The snapshot further contains the online training-admission rate estimate $\hat{p}_t$, the configured sample-injection function $f(\cdot)$, the sandbox records P_t for trajectories whose sandboxes are starting, the ready queues and their instance assignments, the in-flight prefill set Q_t^P with each request's prompt length, instance assignment, and estimated finish time, the active decode set Q_t^D with its current KV-cache footprint K_t^{D}, and historical duration statistics for sandbox startup, tool execution, and reward computation.

Output. The predictor outputs the prefill demand forecast $\hat{W}_{t,S}^{\mathrm{P}}$ and the decode-admission token forecast $\hat{A}_{t,S}^{\mathrm{D}}$ over $[t, t+S)$. $\hat{W}_{t,S}^{\mathrm{P}}$ is measured in prompt tokens and follows the source-specific counting rules in §4.1.2. New requests predicted to arrive from the data sampler, sandbox startup, or tool execution before $t+S$ contribute their estimated prompt-token counts. Requests already in the ready queue or Q_t^P contribute their full prompt lengths only if they are predicted to complete prefill before $t+S$. Queued or in-flight requests not expected to complete within the horizon are excluded; the forecast measures prompt-token volume rather than remaining token-processing work. $\hat{A}_{t,S}^{\mathrm{D}}$ aggregates the prompt lengths of requests whose prefill is expected to complete before $t+S$. The allocator combines the prefill forecast with the current decode footprint K_t^{D} and the KV-cache demand $\hat{K}_{t,S}^{\mathrm{D}}$ derived from predicted admissions to select worker roles in §4.2.

B.2 Data Sampler

In asynchronous rollout, reward completion triggers sample injection, so the sampler forecasts how many trajectories will complete reward computation within $[t, t+S)$. Let R_t be the set of trajectories currently in the reward phase. For $i \in R_t$, the probability of completing reward computation within the horizon is

$$q_i^{\mathrm{rew}} = \Pr(T_i^{\mathrm{rew}} < e_i + S \mid T_i^{\mathrm{rew}} > e_i), \tag{2}$$

where e_i is the elapsed reward time and T_i^{rew} follows a task-conditioned empirical distribution of total reward durations. The conditioning accounts for the fact that reward computation has not finished by time t. Here, a trajectory *passes* if it satisfies the criteria for admission to training. The online estimate $\hat{p}_t \in [0,1]$ approximates the fraction of reward-completed trajectories that satisfy these criteria. Using this shared acceptance-rate estimate, we predict the number of newly eligible trajectories as

$$N_{t,S}^{\mathrm{pass}} = \hat{p}_t \sum_{i \in R_t} q_i^{\mathrm{rew}}. \tag{3}$$

When all reward-completed trajectories are eligible for training, $\hat{p}_t = 1$. Under GRPO (Shao et al., 2024), some group members may have completed reward computation by time t. The sum includes only members still in the reward phase, weighted by their probabilities of completing reward computation within the horizon, and is scaled by $\hat{p}_t$. Members still in DECODE or TOOL are excluded, consistent with the one-step assumption.

Rollout systems inject new samples under different policies—replace-on-reward, fixed-rate refresh, or group replenishment—so we abstract the injection rule as a configurable function $f(\cdot)$. The function is evaluated with the current sampler state, including known group-completion information, held fixed. The estimated number of new launches within the horizon is

$$\hat{N}_{t,S}^{\mathrm{samp}} = f\left(N_{t,S}^{\mathrm{pass}}\right). \tag{4}$$

For nonlinear or group-based injection rules, this is a plug-in approximation rather than an exact expectation over all possible completion outcomes. For newly sampled requests, the forecast assumes a prewarm hit and omits cold starts of not-yet-launched samples. Each predicted launch contributes the estimated prompt-token count $\hat{x}$:

$$\hat{W}_{t,S}^{\mathrm{P,samp}} = \hat{N}_{t,S}^{\mathrm{samp}}\, \hat{x}. \tag{5}$$

B.3 Sandbox Startup

A sandbox already starting from the prewarm pool is assumed to issue its first request within the horizon, contributing its prompt-token count x_k directly. Only a sandbox k currently in cold startup

needs a completion-time prediction: with launch time a_k, its first-request arrival time is $a_k + T^{\text{cold}}$, where T^{cold} follows the measured distribution of total cold-start durations. Since the sandbox is still starting at time t, let $e_k = t - a_k$ and condition on its startup duration exceeding e_k:

$$q_k^{\text{cold}} = \Pr(T^{\text{cold}} < e_k + S \mid T^{\text{cold}} > e_k). \tag{6}$$

With P_t^{pw} and P_t^{cold} denoting the sets of prewarm-hit and cold-start sandboxes in P_t, the expected contribution is

$$\hat{W}_{t,S}^{\text{P,start}} = \sum_{k \in P_t^{\text{pw}}} x_k + \sum_{k \in P_t^{\text{cold}}} q_k^{\text{cold}}\, x_k. \tag{7}$$

Cold-start histories are conditioned on image, node class, and concurrent startup load.

B.4 Ready Queue

Let $g(x_j)$ denote the profiled prefill execution time for prompt length x_j under the selected configuration. We estimate completion times using a request-level serial approximation within each prefill instance, while modeling different instances in parallel. Routing and queue order follow the current scheduling policy; the approximation does not explicitly simulate batched or chunked execution. Let u_j be the observed or predicted request-arrival time, with $u_j = t$ for requests already in the ready queue. For future arrivals, u_j includes the estimated residual reward or sandbox-startup delay, any subsequent sample-injection and prewarmed-startup delay, or the tool-return delay, as applicable. Residual reward and cold-start estimates are conditioned on the operation still being unfinished at t, as in the preceding subsections. Let v_m denote the next available time of prefill instance m, initialized to the latest estimated finish time of its in-flight requests, or t if it is idle. For each not-yet-started request j assigned to instance $m(j)$, the estimated prefill finish time is

$$\hat{b}_j = \max\{u_j, v_{m(j)}\} + g(x_j), \qquad v_{m(j)} \leftarrow \hat{b}_j. \tag{8}$$

This recurrence is evaluated in scheduling order on each instance for queued requests and predicted future arrivals. It prevents a future request from beginning prefill before it arrives and avoids serializing work assigned to different instances. Only requests with $\hat{b}_j < t + S$ are predicted to complete Prefill within the horizon, so the ready-queue contribution is

$$\hat{W}_{t,S}^{\text{P,ready}} = \sum_{j \in \text{Ready},\, \hat{b}_j < t+S} x_j. \tag{9}$$

B.5 Tool Execution

We forecast tool-induced prefill arrivals only for trajectories already in Tool. For trajectory i, let τ_i be the exponential moving average (EMA) of its tool-execution duration and $\hat{\tau}_i$ the most recently observed duration in the same conditioning class; the update is

$$\tau_i \leftarrow (1 - \alpha)\tau_i + \alpha\hat{\tau}_i, \qquad \alpha = 0.1. \tag{10}$$

The EMA is conditioned on task type, tool type, turn index, and concurrent tool load whenever enough samples are available. Let e_i be the elapsed time in the current Tool state; the estimated residual time to the next prefill arrival is

$$r_i = \max(0, \tau_i - e_i). \tag{11}$$

Only trajectories whose tools are predicted to finish within the horizon, i.e., $r_i < S$, contribute, so the predicted tool-induced prefill workload is

$$\hat{W}_{t,S}^{\text{P,tool}} = \sum_{i \in \text{Tool},\, r_i < S} \hat{x}_i, \tag{12}$$

where $\hat{x}_i$ is the estimated prompt length after the tool result is appended.

B.6 In-flight Prefill Requests

Requests already in Prefill are observed directly. For these requests, $\hat{b}_j$ is the finish-time estimate from the runtime snapshot, which accounts for execution progress before t. Their completion times are not recomputed as if prefill were starting from scratch. Each request predicted to complete prefill within the horizon contributes its full prompt length:

$$W_t^{\mathrm{P,inflight}} = \sum_{j \in Q_t^P,\, \hat{b}_j < t+S} x_j. \tag{13}$$

Execution progress affects the finish-time estimate, but does not reduce the prompt length used in the workload forecast. The same finish-time estimates initialize instance availability in Equation 8 and determine decode admissions below.

B.7 Decode Admissions from Prefill Completion

We estimate the KV-cache footprint of newly admitted requests from their prompt lengths, without predicting output lengths. Let $R_{t,S}^P$ contain the in-flight prefills, requests already in the ready queue, and the one-step future arrivals represented in the forecast. For in-flight requests, use the runtime finish-time estimates $\hat{b}_j$; for all other requests, use Equation 8, including their arrival times and the availability of their assigned instances. Let $\kappa^{\mathrm{D}}(x_j)$ denote the KV-cache footprint of a request with prompt length x_j at decode admission under the selected decode configuration. Only requests with $\hat{b}_j < t + S$ contribute to the admission forecasts:

$$\begin{aligned} \hat{A}_{t,S}^{\mathrm{D}} &= \sum_{j \in R_{t,S}^P,\, \hat{b}_j < t+S} x_j, \\ \hat{K}_{t,S}^{\mathrm{D}} &= \sum_{j \in R_{t,S}^P,\, \hat{b}_j < t+S} \kappa^{\mathrm{D}}(x_j). \end{aligned} \tag{14}$$

B.8 Aggregation and Online Correction

The complete prefill forecast is

$$\hat{W}_{t,S}^{\mathrm{P}} = \hat{W}_{t,S}^{\mathrm{P,samp}} + \hat{W}_{t,S}^{\mathrm{P,start}} + \hat{W}_{t,S}^{\mathrm{P,ready}} + \hat{W}_{t,S}^{\mathrm{P,tool}} + W_t^{\mathrm{P,inflight}}. \tag{15}$$

The forecast is refreshed every S seconds using the current allocation. Observed training-admission outcomes update $\hat{p}_t$; arrivals and completions update the reward and startup distributions and the tool-duration EMA, while forecast residuals update a multiplicative bias correction.

C Role Switch

To implement the target allocation from §4.2, the executor selects the least-loaded eligible instance for each required role change. The instance is removed from the routing pool and stops accepting new requests. Affected requests are cancelled, with their input and already generated tokens retained and the KV cache state persisted to remote storage. The rollout controller retries these requests on other instances using the retained prefixes and persisted KV cache, avoiding recomputation of cached tokens. Retries continue generation from the tokens already produced rather than restarting from the original input, preserving the output already visible to the agent. This cancel-and-resend mechanism is transparent to the agent sandbox and requires no intervention from it. Once in-flight work has quiesced, the engine rebuilds role-specific communication and cache resources for the target role, coordinating the transition across all participating ranks. Model weights remain resident in GPU memory throughout execution, so no additional weight loading is required during role switching. The whole role-switch procedure takes around 5 seconds on average.