TAILSIEVE: PARTIAL-ROLLOUT-GUIDED TAIL ROUTING FOR LLM ROLLOUTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large-scale rollouts have become a core component of modern LLM systems, spanning reinforcement learning (RL) post-training, on-policy distillation, and sampling-heavy evaluation. Unlike online serving, which typically targets request-level latency and throughput, synchronous rollouts must collect a complete batch before proceeding. A few long-tail generations can therefore dominate an entire rollout step, and uniform routing can slow these generations further by placing them in high-concurrency decoding batches.

We present TAILSIEVE, a partial-rollout-guided framework that jointly controls tail routing and replica allocation. Motivated by the observed persistence of prompt-level tail tendencies across policy updates, TAILSIEVE uses partial-rollout completion status as a training-free signal for identifying candidate tail groups. Selected prompts are combined with newly sampled prompts, and all responses are generated from scratch under the current policy. A hierarchical controller uses response-work history and a serving model: a fast inner loop adjusts the number of isolated groups to balance completion times, while a slower outer loop adapts how many replicas serve these groups. The resulting low-concurrency execution also enables route-specialized speculative decoding with MTP or DFlash. Across the evaluated configurations, TAILSIEVE achieves up to 1.67× speedup with routing alone and 2.59× with routing and speculation combined, both over uniform group routing.

1 INTRODUCTION

Large-scale rollouts have become a fundamental component of modern large language model (LLM) pipelines. Reinforcement learning with verifiable rewards (RLVR) relies on repeatedly sampling responses and evaluating them with objective reward signals (Guo et al., 2025), while on-policy distillation (OPD) trains a student model on trajectories sampled from its own policy and supervised by a teacher model (Agarwal et al., 2024). Large-scale rollout generation is also widely used for synthetic data generation and sampling-intensive evaluation, such as pass@k and best-of-N evaluation (Wang et al., 2023; Chen et al., 2021). Across these settings, hundreds or thousands of responses may be generated in each optimization step, making rollout generation a dominant component of end-to-end RL training time (Zhou et al., 2025).

Unlike online serving systems, where the primary objectives are request-level latency, throughput, and cost efficiency (Qiu et al., 2024; Liu et al., 2026; Yuan et al., 2026; Hu et al., 2026), synchronous rollout pipelines are governed by a different performance objective: step-level makespan. The optimization process cannot proceed until a required set of generations is completed. Therefore, the completion time of a rollout step is determined by the slowest unfinished requests (Figure 1(b)). Since LLM response lengths exhibit strong skewness, a small number of extremely long generations can dominate the overall execution time and stall the entire rollout process (Zhou et al., 2025; Zhong et al., 2025).

Existing works improve rollout efficiency by reducing synchronization stalls, reorganizing long-tail workloads, or accelerating decoding (Zhou et al., 2025; Gao et al., 2026; Zhong et al., 2025; Qin et al., 2026; Shao et al., 2026; Xu et al., 2026). We further discuss these works in Appendix H. Here, we focus on a fundamental routing problem in synchronous rollouts: **How should requests be distributed across replicas to minimize step-level makespan?**

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

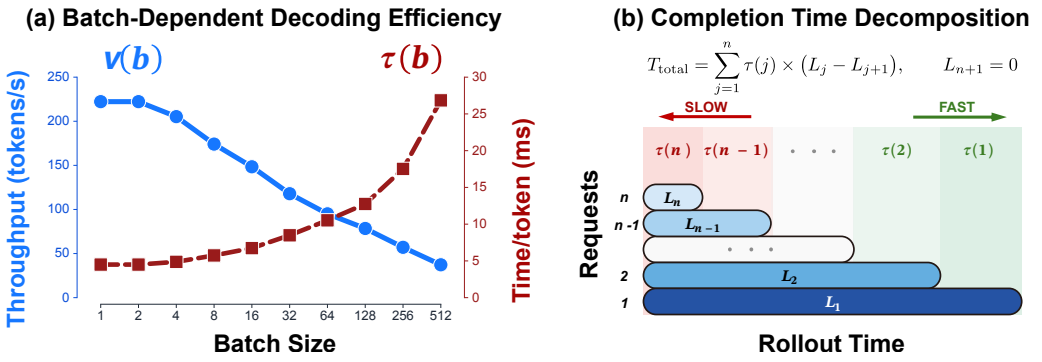


Figure 1: Measured per-request decode throughput versus active batch size (a) and the resulting replica-time decomposition (b). The left axis reports tokens/s per request, and the right axis reports ms/token. The objective follows the changing concurrency until the last request finishes.

To answer this question, we begin by studying the offline optimal assignment, assuming that all response lengths are known in advance. Our analysis shows that the optimum follows a *tail-isolation-with-balancing* structure: it assigns a small number of extreme-tail requests to one replica, which we call the tail replica, while routing most requests to the other, or bulk replica (Figure 2(a)). We further find a simpler and more tractable solution: isolating the k longest responses on the tail replica and routing the rest to the bulk replica can closely match the optimal makespan (Figure 2(b); Appendix B.4).

Applying this top-k strategy online to reduce rollout makespan requires identifying likely tail prompts in advance. The gradual evolution of the policy during RL training suggests a possible way forward: a prompt's behavior in one rollout may help identify whether it will generate long responses in the next. Motivated by this possibility, we examine response lengths across checkpoints. We find that prompt identity accounts for most of the variation (Appendix C.3): prompts that generate long responses tend to remain near the tail after policy updates (Figure C.1). We also find that groups unfinished at a partial-rollout cutoff capture most of the longest-response groups (Figure 3).

These observations motivate TAILSIEVE's use of partial-rollout status as a prompt-level tail signal. It routes unfinished prompt groups—likely to be the top-k longest groups—to dedicated tail replicas in the next round, regenerating their responses under the current policy (Figure 5). TAILSIEVE requires no auxiliary length model and does not carry unfinished trajectories across policy updates. An online controller jointly adapts tail workload and replica capacity (Section 3.1), while each update mixes regenerated tail prompts with newly sampled prompts. Empirically, accepted-group mean-length distributions remain close to those under uniform routing (Figure F.1).

Our contributions are summarized as follows:

(1) *Long-Tail-Aware Routing Principles*: We formulate long-tail-aware rollout routing as a step-level makespan minimization problem. In the long-tail regime, the offline optimum follows **tail isolation with load balancing**, and a simple top-k policy closely approximates the exact oracle (Section 2; Figure 2).

(2) *Adaptive Hierarchical Routing*: We develop TAILSIEVE, which uses partial rollout as a training-free prompt-level tail signal. Its hierarchical controller jointly adapts the isolated workload and replica split using pool completion times, response-work history, and measured decoding throughput (Section 3.1; Algorithm 1).

(3) *Tail Isolation Amplifies Speculative Decoding*: We show that tail isolation amplifies the gains from speculative decoding by concentrating long responses on low-concurrency tail replicas. This structure enables **deeper speculation** and extends its beneficial operating range, which TAILSIEVE exploits through route-specialized proposal depths to further reduce rollout makespan (Figure 6(a)).

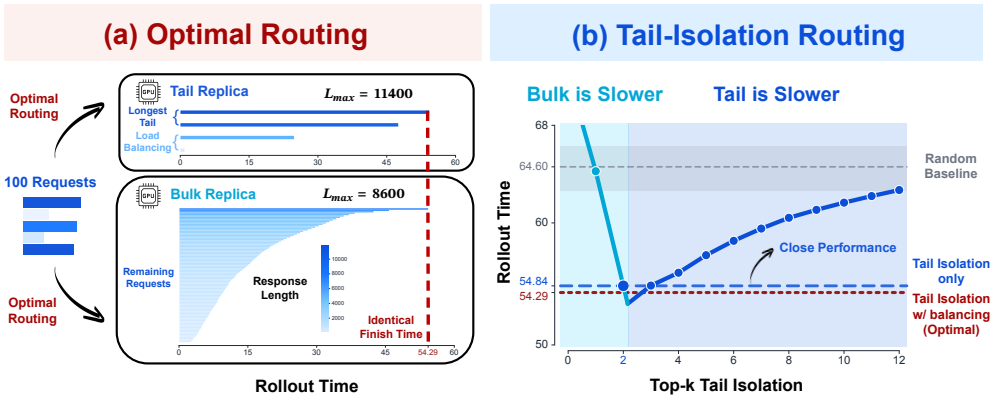


Figure 2: A representative offline assignment and top-k isolation sweep from the original rollout analysis. The oracle uses few tail requests plus balancing traffic; top-k is a simpler restricted policy.

2 Routing under Concurrency-Dependent Decoding

We begin with a simple question: **if we knew which generations were going to be long, how should they be routed across replicas?**

A decoding-time objective. We first consider a simple model that focuses only on decoding. Let $v(b)$ be the per-request throughput at active batch size b, and let $\tau(b) = 1/v(b)$. For a replica with request lengths $L_1 \geq \cdots \geq L_n$ and $L_{n+1} = 0$, its completion time is

$$T(S) = \sum_{j=1}^{n} \tau(j)(L_j - L_{j+1}) = \sum_{j=1}^{n} L_j[\tau(j) - \tau(j-1)], \qquad \tau(0) = 0. \tag{1}$$

The routing objective is to minimize the maximum completion time across replicas. Figure 1 illustrates the decoding model.

Exact oracle and tail isolation. We process requests from longest to shortest. Assigning a request of length L_i to a replica with c existing requests increases its completion time by $L_i[\tau(c+1) - \tau(c)]$. A dynamic program tracks the completion times of the two replicas and prunes dominated states. Appendix B.3 describes the oracle computation and explains why the dynamic program is exact.

In the representative 100-request case (Figure 2), the optimal assignment is highly imbalanced: the tail lane contains only four responses—the two longest responses together with two additional responses for load balancing—yet takes nearly the same time to finish as the bulk lane with the remaining 96 responses. We further discovered a much more tractable approximation: simply isolating the k longest responses already closely matches the optimal makespan (Figure 2b). Across 100 sampled cases, the best-k solution is within 4% of the optimum in all cases and within 1% in 72% of cases (Appendix B.4).

2.1 Partial Rollout as a Cross-Round Tail Signal

Although completion lengths are unknown online, the top-k isolation result provides a useful insight: we only need to identify the longest responses in each batch, rather than predict their exact lengths.

Unlike prior systems that carry or reuse unfinished trajectories across rollout steps (Zhou et al., 2025; Kimi Team, 2025), TailSieve uses cutoff-time status only as a prompt-level routing signal. Figure 3 evaluates this signal at prompt-group granularity, comparing groups unfinished at cutoffs with the final group-length ranking. We further conduct a cross-checkpoint study and find that, even as the model evolves during training, prompt-level length rankings remain highly informative for tail identification: adjacent-checkpoint AUC ranges from 0.957 to 0.985. Our cross-checkpoint study finds that prompt identity accounts for most of the response-length variance (Appendix C).

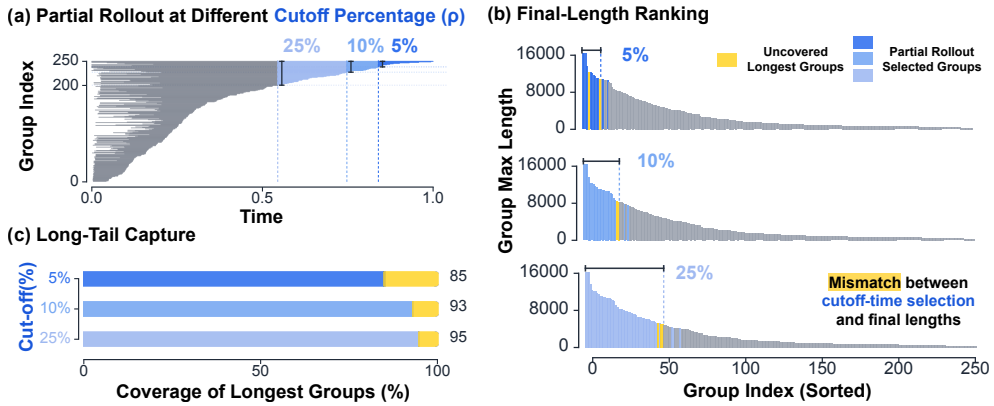


Figure 3: Partial-rollout tail identification at prompt-group granularity. The cutoff parameter ρ is the number of excess groups divided by the completion quota. Appendix C.1 defines the virtual cutoff, group-length ranking, and coverage calculation.

2.2 Coupled Tail-Workload and Replica Allocation

Our remaining challenge is deciding how much tail workload should be isolated. Intuitively, when a small number of responses are substantially longer than the rest, only these extreme responses should be routed separately; when response lengths are nearly uniform, tail isolation provides little benefit. As the response-length distribution shifts during training, the optimal operating point therefore shifts as well. Figure 4 illustrates this relationship between workload shape and the preferred tail configuration. We provide a detailed analysis in Appendix B.2 showing that the optimal operating point depends jointly on the response-length distribution and the concurrency-dependent decoding throughput.

This workload-dependent operating point motivates TailSieve: rather than using a fixed tail fraction or replica split, TailSieve adapts both the amount of isolated tail workload and the resources assigned to it according to the current response-length distribution. The request-level oracle further reveals that the key structural benefit comes from isolating long-tail work. TailSieve translates this oracle insight into an online, deployable mechanism by identifying tail-prone prompt groups and routing them to dedicated replicas.

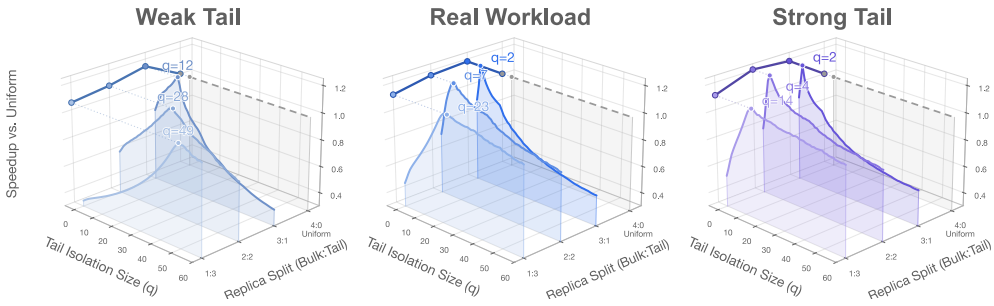


Figure 4: Joint-control landscapes showing that the optimal isolation size and replica split vary with tail strength. Strengthening the response-length tail moves the preferred operating point toward a smaller isolation size q, whereas weakening the tail requires a larger q to keep the two pools balanced and brings the optimum closer to uniform routing.

3 TailSieve Design

A mixed tail–bulk pipeline. TailSieve converts the routing structure of Section 2 into a cross-round pipeline. A partial rollout identifies unfinished prompt groups as tail candidates. In the next

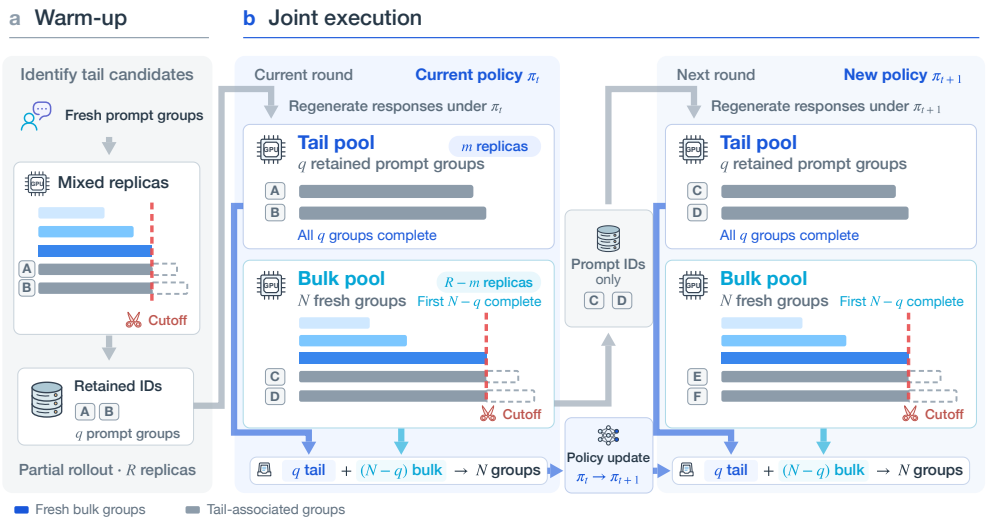


Figure 5: TAILSIEVE warm-up and joint execution. Unfinished prompt identities enter the tail pool and regenerate under the current policy.

step, the tail pool regenerates these prompts under the current policy, while the bulk pool processes fresh groups. Only prompt identities are carried across steps.

At a stable allocation, the tail pool uses m_t of R replicas to regenerate q_t carried groups; bulk launches N fresh groups on the rest. The update accepts all q_t tail groups and the first $N - q_t$ completed bulk groups, totaling N. Unaccepted bulk groups form the next carry. Thus q_t controls isolated workload and m_t tail capacity; $(q_t, m_t) = (0, 0)$ gives uniform routing.

Fresh regeneration ensures that all responses are sampled under the current policy. Under the idealized assumptions in Appendix G, the mixed tail–bulk batch preserves the response-length distribution in expectation for a fixed policy. Empirically, the accepted-group mean-length distributions remain close to those under uniform routing (Appendix F.3). Overall, TAILSIEVE exploits persistent prompt-level tail signals for efficient routing while keeping every response on-policy.

3.1 HIERARCHICAL ALLOCATION

For each rollout step, execution completes only when both the tail and bulk pools finish, so the slower pool determines the step time:

$$T_t(q, m) = \max\left\{T_t^{\text{tail}}(q, m), T_t^{\text{bulk}}(N - q, R - m)\right\}. \tag{2}$$

Here, $T_t^{\text{bulk}}(N - q, R - m)$ denotes the time to collect $N - q$ complete groups from the N groups submitted to bulk. A fast inner loop adjusts the tail quota q from measured completion times, while a slower outer loop uses a profiled throughput curve and the empirical completed-token work profile to update the replica split m.

Inner loop: balance completion times. For a fixed replica split, the min–max objective is minimized when both pools finish together (Appendix E.1). Eligible rounds give the group-processing rates

$$T^{\text{tail}} \approx \frac{q}{r^{\text{tail}}}, \qquad T^{\text{bulk}} \approx \frac{N - q}{r^{\text{bulk}}}. \tag{3}$$

Equating the two gives the balanced quota

$$q_{\text{balanced}} = N \frac{\bar{r}^{\text{tail}}}{\bar{r}^{\text{tail}} + \bar{r}^{\text{bulk}}}, \tag{4}$$

where $\bar{r}$ is an exponential moving average. The controller projects and clips this target, then selects the better of its floor and ceiling under the same min–max surrogate. The core rule stops when the

Algorithm 1: TAILSIEVE's hierarchical allocation controller.

Input: Groups N, replicas R, feasible quotas $\mathcal{Q}$, initial (q, m), fixed throughput curve G, inner and outer thresholds ϵ_q, ϵ_m

for *each outer-loop update* **do**
  for *each inner-loop update at fixed* m **do**
    Update the empirical work profile $\mathcal{L}$ with accepted complete-group token counts
    Observe $T_t^{\text{tail}}, T_t^{\text{bulk}}$ and set $r_t^{\text{tail}} \leftarrow q/T_t^{\text{tail}}$, $r_t^{\text{bulk}} \leftarrow (N-q)/T_t^{\text{bulk}}$
    $\bar{r}_t^s \leftarrow \beta \bar{r}_{t-1}^s + (1-\beta) r_t^s$, $s \in \{\text{tail}, \text{bulk}\}$
    if *mean normalized gap* $\le \epsilon_q$ **then break**
    $q_{\text{balanced}} \leftarrow \text{clip}_{[\min \mathcal{Q}, \max \mathcal{Q}]}\big(N\bar{r}_t^{\text{tail}}/(\bar{r}_t^{\text{tail}} + \bar{r}_t^{\text{bulk}})\big)$
    $q^+ \leftarrow \arg\min_{q' \in \{\lfloor q_{\text{balanced}} \rfloor, \lceil q_{\text{balanced}} \rceil\}} \max\{q'/\bar{r}_t^{\text{tail}}, (N-q')/\bar{r}_t^{\text{bulk}}\}$
    if $q^+ \neq q$ **then** Apply q^+; restart the EMA and fresh stability evidence
  end
  Predict $\widehat{T}^{\text{tail}}, \widehat{T}^{\text{bulk}}$ from fixed G and $\mathcal{L}$
  $\mathcal{N}(m) \leftarrow (\{m\} \cup \{m-1, m+1\}) \cap \{1, \ldots, R-1\}$
  $(m^+, q^+) \leftarrow \arg\min_{m' \in \mathcal{N}(m),\, q' \in \mathcal{Q}} \max\{\widehat{T}^{\text{tail}}(q', m'), \widehat{T}^{\text{bulk}}(q', m')\}$
  $\widehat{T}_{\text{current}} \leftarrow \min_{q' \in \mathcal{Q}} \max\{\widehat{T}^{\text{tail}}(q', m), \widehat{T}^{\text{bulk}}(q', m)\}$
  $\text{gain}_{\text{pred}} \leftarrow (\widehat{T}_{\text{current}} - \max\{\widehat{T}^{\text{tail}}(q^+, m^+), \widehat{T}^{\text{bulk}}(q^+, m^+)\})/\widehat{T}_{\text{current}}$
  if $m^+ = m$ *or* $\text{gain}_{\text{pred}} \le \epsilon_m$ **then return** (q, m)
  $(m, q) \leftarrow (m^+, q^+)$; reset the inner-loop state
end

mean gap is within ϵ_q; Appendix D gives the measurement validity checks, finite-integer safeguards, and parameter settings used in the implementation.

Outer loop: adjust replica capacity. After balancing q, a throughput curve and the empirical work profile score the current and feasible adjacent splits after rebalancing their quota:

$$\widehat{\Phi}(m') = \min_{q' \in \mathcal{Q}} \max\left\{\widehat{T}^{\text{tail}}(q', m'), \widehat{T}^{\text{bulk}}(q', m')\right\}. \tag{5}$$

The controller moves one replica only when the best neighbor improves this score by more than ϵ_m, without timing candidate layouts online. Appendix D gives the implementation; Appendix E.3 gives the fluid-limit support for neighboring-split search.

Increasing q adds fresh tail groups while bulk still submits N fresh groups. Transition rounds do not update control measurements; quota decreases reassign excess carry. Appendix G gives the inventory and settlement conditions.

3.2 ROUTE-SPECIALIZED SPECULATIVE DECODING

Tail isolation creates a low-concurrency stream with a long decoding horizon. It permits deeper speculative proposals on the tail route, while the bulk route uses a conservative depth to limit verification cost. We instantiate this separation with native MTP and DFlash (Qwen Team, 2026; Chen et al., 2026); target-model verification preserves the target distribution (Leviathan et al., 2023). Routing and speculation are complementary: routing controls the batches in which verification executes, and each route can choose its backend and depth. We also explored suffix decoding (Oliaro et al., 2025), but limited accepted lengths and ragged verification shapes that hinder full-batch CUDA-graph replay outweighed its benefit; it is disabled by default.

4 EVALUATION

4.1 EXPERIMENTAL SETUP

We evaluate TAILSIEVE on five dense and mixture-of-experts model configurations: Qwen3.5-35B-A3B, Qwen3.5-4B, and Qwen3.5-2B (Qwen Team, 2026), together with Qwen3-30B-A3B-Instruct-2507 and Qwen3-4B-Instruct-2507 (Yang et al., 2025).

All performance and RL experiments use vLLM on the same server with eight H100 GPUs (Kwon et al., 2023). Large models use four TP2 replicas, while the 4B and 2B models use eight TP1 replicas. Step-wise experiments use mathematical prompts from DeepScaleR (Agentica Team, 2025) and coding prompts from KodCode-Light-RL-10K (Xu et al., 2025); the RL experiment uses the mathematical workload. Each prompt group contains eight responses and the default rollout contains 64 groups. Paired runs share the prompt workload, sampling configuration, and total replica budget. Appendix A.1 gives the complete settings.

4.2 Step-wise Rollout Evaluation

Routing alone reaches 1.67× speedup over uniform routing. We evaluate TailSieve after the joint allocation (q, m) has converged and the system has entered steady-state cross-round execution. We report generation time at the selected settled operating point, averaged over three runs with different random seeds. All selected requests are regenerated from scratch under the current policy: no partial responses, prefix tokens, or cross-round KV-cache state are reused. The accepted-group mean-length distributions remain close to those under uniform routing; Appendix F.3 reports the corresponding empirical CDFs.

We first disable speculative decoding and compare TailSieve with the routing and scheduling baselines under the same requests, model configuration, and total replica budget. Table 1 summarizes the routing-only results across both workloads. TailSieve achieves up to 1.67× speedup, while the fixed-allocation variant shows the additional benefit of adapting the replica split.

Table 1: Routing-only speedup over uniform group routing.

DeepScaleR

Routing policy	Qwen3.5-35B-A3B	Qwen3.5-4B	Qwen3.5-2B	Qwen3-30B-A3B	Qwen3-4B
Uniform	1.000×	1.000×	1.000×	1.000×	1.000×
Group LPT*	1.123×	1.151×	1.154×	1.085×	1.225×
StreamRL-style Oracle*	1.030×	1.113×	1.136×	0.928×	1.174×
Seer-style Oracle*	1.189×	1.162×	1.169×	1.082×	**1.358×**
TailSieve (Fixed Replica Allocation)	1.219×	1.199×	1.144×	**1.112×**	1.186×
TailSieve	**1.346×**	**1.233×**	**1.180×**	**1.112×**	1.254×

KodCode-10K

Routing policy	Qwen3.5-35B-A3B	Qwen3.5-4B	Qwen3.5-2B	Qwen3-30B-A3B	Qwen3-4B
Uniform	1.000×	1.000×	1.000×	1.000×	1.000×
Group LPT*	1.033×	1.001×	1.106×	1.083×	1.107×
StreamRL-style Oracle*	1.167×	1.033×	1.100×	1.094×	0.992×
Seer-style Oracle*	1.282×	0.971×	1.150×	1.062×	1.406×
TailSieve (Fixed Replica Allocation)	1.560×	1.310×	1.143×	1.166×	1.179×
TailSieve	**1.670×**	**1.403×**	**1.214×**	**1.342×**	**1.442×**

*Length-aware baselines use known historical group-average lengths.

Uniform routing distributes complete groups evenly by count. Before routing the evaluated round, the marked length-aware baselines are given advance length estimates: each prompt group's average completion length from a matching historical run. Group LPT retains group-level placement, while StreamRL-style Oracle (Zhong et al., 2025) and Seer-style Oracle (Qin et al., 2026) assign requests individually. We select these two routing-only adaptations to compare length-aware pool partitioning and request-level load balancing on the same vLLM executor. The fixed-allocation variant ablates TailSieve's outer replica loop. Appendix A.2 specifies each policy. Appendix H discusses how this scope differs from systems that reorganize rollout rounds, reuse cross-policy trajectories, or accelerate decoding.

Routing gains persist as rollout size increases. With speculation disabled and the replica budget fixed, we increase the rollout size from 64 to 128 and 256 prompt groups, each with eight responses. Routing remains faster than uniform routing across all five models and both workloads, retaining 1.09–1.31× speedup at 256 groups (Table 2). Appendix A.2 gives the comparison protocol.

Table 2: Routing-only speedup over uniform routing at increasing rollout group counts N.

Model	DeepScaleR			KodCode-10K		
	$N = 64$	$N = 128$	$N = 256$	$N = 64$	$N = 128$	$N = 256$
Qwen3.5-35B-A3B	1.346×	1.199×	1.122×	1.670×	1.214×	1.142×
Qwen3.5-4B	1.233×	1.225×	1.191×	1.403×	1.453×	1.240×
Qwen3.5-2B	1.180×	1.129×	1.177×	1.214×	1.108×	1.101×
Qwen3-30B-A3B	1.112×	1.023×	1.121×	1.342×	1.163×	1.091×
Qwen3-4B	1.254×	1.215×	1.306×	1.442×	1.367×	1.248×

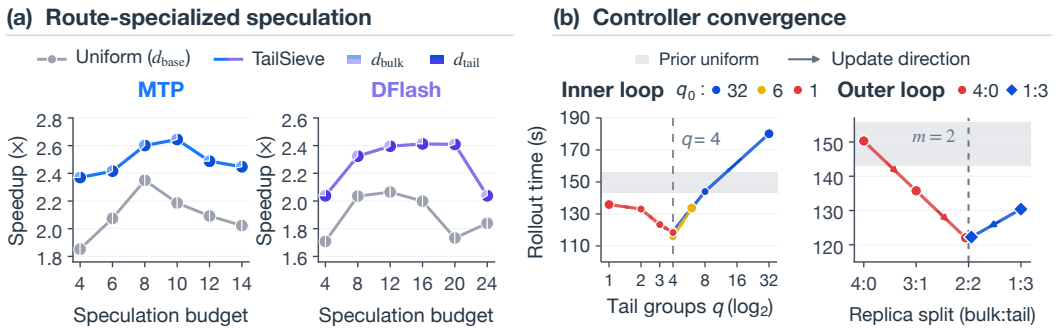


Figure 6: Route-specialized speculative decoding **(a)** and controller convergence **(b)** on Qwen3.5-35B-A3B. Marker sectors in (a) denote bulk and tail proposal depths. Panel (b) shows inner- and outer-loop convergence in separate experiments.

Routing and speculation jointly reach 2.59× speedup. We next combine TAILSIEVE routing with speculative decoding. MTP uses three speculative tokens, while DFlash uses a block size of four. Routing and speculation together reach 2.59× speedup over uniform routing without speculative decoding on the coding workload with Qwen3.5-35B-A3B (Table 3).

Table 3: Step-wise speedup of TAILSIEVE with fixed replica split m and adaptive quota q, using routing alone or with MTP or DFlash, relative to uniform routing without speculative decoding.

Model	DeepScaleR			KodCode-10K		
	Routing	+MTP	+DFlash	Routing	+MTP	+DFlash
Qwen3.5-35B-A3B	1.219×	2.390×	2.181×	1.560×	2.586×	2.378×
Qwen3.5-4B	1.199×	2.414×	2.091×	1.310×	2.408×	2.148×
Qwen3.5-2B	1.144×	2.161×	–	1.143×	2.349×	–

– indicates that no official DFlash implementation is available for Qwen3.5-2B. Appendix F.1 reports the corresponding times and same-speculation controls.

Tail isolation improves speculation and extends its useful depth range. Figure 6(a) compares MTP and DFlash on Qwen3.5-35B-A3B with matched total proposal-depth budgets. With d_{base} denoting the depth on each baseline replica, uniform routing uses $(d_{\text{base}}, d_{\text{base}})$, while TAILSIEVE uses route-specific depths satisfying $d_{\text{bulk}} + d_{\text{tail}} = 2d_{\text{base}}$. For example, MTP-4/MTP-4 is compared with MTP-3 on the high-concurrency bulk route and MTP-5 on the low-concurrency tail route. Across the evaluated budgets, route-specialized speculation consistently outperforms uniform routing and shifts the best speculation budget upward: the isolated tail remains effective at larger depths where uniform routing no longer improves.

Both control loops converge and reduce rollout time. Across different initializations, the inner-loop quota updates and outer-loop replica-allocation decisions converge to faster configurations (Figure 6(b)). Figure D.1 shows the round-by-round inner-loop convergence.

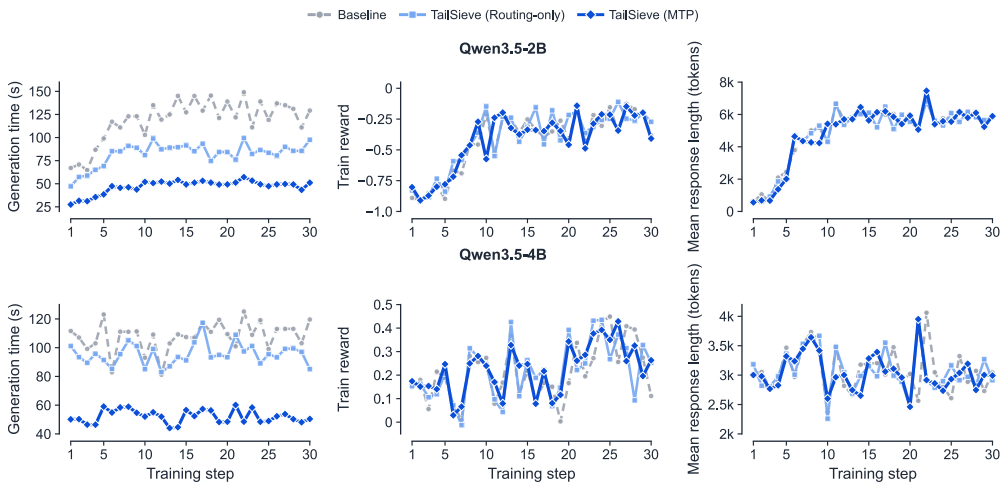


Figure 7: GRPO results for Qwen3.5-2B (top) and Qwen3.5-4B (bottom): generation time, reward, and mean response length.

Table 4: Downstream accuracy (AIME: mean over three evaluation seeds); parentheses report percentage-point changes from the corresponding base model.

Model	Method	GSM8K pass@1	MATH-500 pass@1	AIME24 pass@10	AIME25 pass@10
Qwen3.5-2B	Base model	9.40%	13.40%	33.33%	26.67%
	Baseline	75.97% (+66.57)	76.20% (+62.80)	56.67% (+23.33)	36.67% (+10.00)
	TailSieve (Routing-only)	76.57% (+67.17)	74.60% (+61.20)	57.78% (+24.44)	41.11% (+14.44)
	TailSieve (MTP)	76.42% (+67.02)	74.40% (+61.00)	53.33% (+20.00)	36.67% (+10.00)
Qwen3.5-4B	Base model	89.61%	84.40%	90.00%	82.22%
	Baseline	89.16% (−0.45)	88.80% (+4.40)	91.11% (+1.11)	80.00% (−2.22)
	TailSieve (Routing-only)	89.46% (−0.15)	87.40% (+3.00)	91.11% (+1.11)	82.22% (+0.00)
	TailSieve (MTP)	89.61% (+0.00)	86.60% (+2.20)	90.00% (+0.00)	84.44% (+2.22)

4.3 End-to-End RL Evaluation

Rollout acceleration persists during RL training. Speculation depths are predefined for the RL runs. We integrate TailSieve into GRPO training (Shao et al., 2024) and measure the complete rollout-generation stage at each training step, including routing and controller execution.

Figure 7 shows that TailSieve, with and without MTP, accelerates rollout generation on both models while maintaining comparable reward and mean response length.

Observed downstream accuracy remains comparable. The final checkpoints achieve comparable downstream accuracy (Table 4). We evaluate checkpoint quality on GSM8K and MATH-500 (Cobbe et al., 2021; Lightman et al., 2024), and the 2024 and 2025 AIME problem sets (Mathematical Association of America, 2026), reporting pass@1 for the former and pass@10 for the latter.

5 Conclusion

We presented TailSieve, a partial-rollout-guided framework for mitigating long-tail stalls in synchronous LLM rollouts. Our key insight is that isolating a small amount of tail workload can reduce step makespan, but the optimal degree of isolation depends on the evolving workload and decoding concurrency. TailSieve turns this insight into an online system by using partial rollouts to identify tail-prone prompt groups and jointly adapting tail workload and replica capacity. Across evaluated models and workloads, TailSieve achieves up to 1.67× routing-only speedup and 2.59× end-to-end speedup with route-specialized speculative decoding, while maintaining comparable training quality. We hope this work motivates more adaptive rollout systems that exploit workload structure to improve the efficiency of large-scale LLM training.

AI USE STATEMENT

Generative AI tools were used to assist with language editing, improve clarity, and polish the presentation of this manuscript.

REPRODUCIBILITY STATEMENT

The decoding model and controller are specified in Sections 2 and 3.1. Appendix D gives the controller parameters and executable rules, Appendix A records the experimental configurations, and Appendix B.3 gives the oracle recurrence and exactness argument. Appendix G states the assumptions and limits of the distribution analysis.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=3zKtaqxLhW.

Agentica Team. DeepScaleR-Preview-Dataset. Hugging Face Dataset, 2025. URL https://huggingface.co/datasets/agentica-org/DeepScaleR-Preview-Dataset.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, Tongkai Yang, Binhang Yuan, and Yi Wu. AReaL: A large-scale asynchronous reinforcement learning system for language reasoning. In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc., 2025. doi: 10.52202/085713-1218.

Wei Gao, Yuheng Zhao, Dakai An, Tianyuan Wu, Lunxi Cao, Shaopan Xiong, Ju Huang, Weixun Wang, Siran Yang, Wenbo Su, Jiamang Wang, Lin Qu, Bo Zheng, and Wei Wang. RollPacker: Taming long-tail rollouts for RL post-training with tail batching. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pp. 849–866, Renton, WA, May 2026. USENIX Association. URL https://www.usenix.org/conference/nsdi26/presentation/gao-wei.

Daya Guo, Dejian Yang, Haowei Zhang, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645:633–638, 2025. doi: 10.1038/s41586-025-09422-z.

Jingkai He, Tianjian Li, Erhu Feng, Dong Du, Qian Liu, Tao Liu, Yubin Xia, and Haibo Chen. History doesn't repeat itself but rollouts rhyme: Accelerating reinforcement learning with RhymeRL. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '26, pp. 929–945, New York, NY, USA, 2026. Association for Computing Machinery. doi: 10.1145/3779212.3790172.

Xinyi Hu, Yuhao Shen, Baolin Zhang, Hengxin Zhang, Jun Dai, Shuang Ge, Lei Chen, Yue Li, and Mingcheng Wan. ECHO: Elastic speculative decoding with sparse gating for high-concurrency scenarios. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Kimi Team. Kimi k1.5: Scaling reinforcement learning with LLMs. *arXiv preprint arXiv:2501.12599*, 2025.

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*, SOSP '23, pp. 611–626, New York, NY, USA, 2023. Association for Computing Machinery. doi: 10.1145/3600006.3613165.

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*, pp. 19274–19286. PMLR, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Yucheng Li, Huiqiang Jiang, Yang Xu, Jianxin Yang, Yi Zhang, Yizhong Cao, Yuhao Shen, Fan Zhou, Rui Men, Jianwei Zhang, An Yang, Bowen Yu, Bo Zheng, Fei Huang, Junyang Lin, Dayiheng Liu, and Jingren Zhou. Breaking entropy bounds: Accelerating RL training via MTP with rejection sampling. *arXiv preprint arXiv:2606.12370*, 2026.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

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*, 2025a.

Xunzhuo Liu, Bowei He, Xue Liu, Andy Luo, Haichen Zhang, and Huamin Chen. Dual-Pool token-budget routing for cost-efficient and reliable LLM serving. *arXiv preprint arXiv:2604.08075*, 2026.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-like training: A critical perspective. In *Conference on Language Modeling*, 2025b. URL https://openreview.net/forum?id=5PAF7PAY2Y.

Mathematical Association of America. MAA invitational competitions: American invitational mathematics examination. Mathematical Association of America, 2026. URL https://maa.org/maa-invitational-competitions/. Accessed 2026-08-10.

Gabriele Oliaro, Zhihao Jia, Daniel Campos, and Aurick Qiao. SuffixDecoding: Extreme speculative decoding for emerging AI applications. In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/b7aea253ab34a773967f1e4cdea9e4fb-Abstract-Conference.html.

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. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*, pp. 883–901, Seattle, WA, July 2026. USENIX Association. URL https://www.usenix.org/conference/osdi26/presentation/qin.

Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew T. Kalbarczyk, Tamer Başar, and Ravishankar K. Iyer. Efficient interactive LLM serving with proxy model-based sequence length prediction. In *The 5th International Workshop on Cloud Intelligence / AIOps at ASPLOS 2024*, volume 5, pp. 1–7, San Diego, CA, USA, April 2024. Association for Computing Machinery. URL https://cloudintelligenceworkshop.org/2024/accepted-papers.html.

Zekai Qu, Yinxu Pan, Ao Sun, Chaojun Xiao, and Xu Han. CoPRIS: Efficient and stable reinforcement learning via concurrency-controlled partial rollout with importance sampling. *arXiv preprint arXiv:2511.05589*, 2025.

Qwen Team. Qwen3.5: Towards native multimodal agents. Qwen Technical Blog, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Zelei Shao, Vikranth Srivatsa, Sanjana Srivastava, Qingyang Wu, Alpay Ariyak, Xiaoxia Wu, Ameen Patel, Jue Wang, Percy Liang, Tri Dao, Ce Zhang, Yiying Zhang, Ben Athiwaratkun, Chenfeng Xu, and Junxiong Wang. Beat the long tail: Distribution-aware speculative decoding for RL training. In *Proceedings of Machine Learning and Systems*, volume 8, 2026. URL https://proceedings.mlsys.org/paper_files/paper/2026/hash/cbc4ab80cd77aa0eb87da062fbcddb46-Abstract-Conference.html.

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.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-Instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13484–13508, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.754.

Huanyi Xie, Yubin Chen, Liangyu Wang, Lijie Hu, and Di Wang. Predicting LLM output length via entropy-guided representations. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=3loQDtveWI.

Yuhang Xu, Kaibin Tian, Yang Tian, Zhice Yang, Yifeng Yu, Yan Li, Shengzhong Liu, Fan Wu, and Guihai Chen. BubbleSpec: Turning long-tail bubbles into speculative rollout drafts for synchronous reinforcement learning. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. KodCode: A diverse, challenging, and verifiable synthetic dataset for coding. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6980–7008, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.365.

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

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/a4277440d50f1f15d2cb4c14f7e0c0d2-Abstract-Conference.html.

Ying Yuan, Pengfei Zuo, Bo Wang, Zhangyu Chen, Zhipeng Tan, and Zhou Yu. DualMap: Enabling both cache affinity and load balancing for distributed LLM serving. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=zCadrJ32Xn.

Haoyu Zheng, Yongqiang Zhang, Fangcheng Fu, Xiaokai Zhou, Hao Luo, Hongchao Zhu, Yuanyuan Zhu, Hao Wang, Xiao Yan, and Jiawei Jiang. Scheduling LLM inference with uncertainty-aware output length predictions. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Yinmin Zhong, Zili Zhang, Xiaoniu Song, Hanpeng Hu, Chao Jin, Bingyang Wu, Nuo Chen, Yukun Chen, Yu Zhou, Changyi Wan, Hongyu Zhou, Yimin Jiang, Yibo Zhu, and Daxin Jiang. StreamRL: Scalable, heterogeneous, and elastic RL for LLMs with disaggregated stream generation. *arXiv preprint arXiv:2504.15930*, 2025. URL https://arxiv.org/abs/2504.15930.

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.

A Experimental Setup and Protocols

A.1 Common Experimental Setup

Experiments run on one node with eight NVIDIA H100 GPUs and use vLLM as the rollout backend (Kwon et al., 2023). Experiments use vLLM `0.23.1rc1.dev562+g3483240b7`, based on main commit `3483240b7` with local Qwen3.5/V2 compatibility changes. These changes cover LM-head access, attention metadata for the draft and target models, and hybrid cache handling. Its Model Runner V2 (MRV2) DFlash path derives from `vllm-project/vllm` PR #44586, including full CUDA-graph support for DFlash. The two large-model configurations use four TP2 replicas, while the 4B and 2B configurations use eight TP1 replicas. The total replica budget remains fixed in every comparison.

The step-wise experiments use mathematical prompts from DeepScaleR (Agentica Team, 2025) and bounded-reasoning coding prompts from `KodCode/KodCode-Light-RL-10K` (Xu et al., 2025). Step-wise results are averaged over three runs with different random seeds. The RL experiments use the mathematical workload. Responses are sampled independently in each run, so realized response lengths and token counts may differ across otherwise matched configurations. Table A.1 reports the sampling settings shared by step-wise evaluation and RL training rollouts.

Table A.1: Rollout sampling configuration shared by the step-wise and end-to-end RL experiments.

Parameter	Setting
Maximum prompt length	512 tokens
Maximum response length	16,384 tokens
Temperature	0.7
Top-p	0.8
Top-k	20
Presence penalty	1.5
Thinking mode	Disabled
Stop sequence	`<END>`

Prompt templates. The DeepScaleR workload uses no system message and applies the following user template, where `{question}` is the dataset problem:

Mathematics — User

```
{question}
Please solve the problem step by step.  End your response with
exactly one final answer in this form:
\boxed{...}<END>
After <END>, stop immediately.  Do not write any words, punctuation,
newline, or explanation after <END>.
```

The coding workload uses the following system and user templates:

Code — System

```
You are an expert Python programmer.
```

Code — User

```
Solve the coding problem below.  Provide your reasoning and
explanation, including the key algorithm, its correctness, important
edge cases, and its time and space complexity.  Then provide exactly
one complete Python function.  After the function, write <END> and
stop immediately.  Do not repeat or revise the answer.

Problem:
{question}
```

```
Required function:
{function_declaration}

Specification:
{docstring}
```

A.2 Routing Baselines and Execution Protocols

Routing-policy implementations. Uniform group routing distributes complete groups evenly by count for the divisible group counts used in the baseline comparisons. All marked length-aware baselines in Table 1 use each prompt group's average completion length from a matching historical run.

Group LPT (historical-length group scheduling). Group LPT (longest processing time first) sorts whole eight-response groups by decreasing historical mean length and greedily assigns each group to the replica with the lowest accumulated predicted load among those below the equal group-count capacity.

StreamRL-style Oracle (historical-length skewness-aware routing). This baseline adapts skewness-aware dispatching (Zhong et al., 2025) at request granularity. Each response inherits its group's historical mean length. We sort the $n = 512$ requests in decreasing order of this estimate, assign the first $\lfloor 0.2n \rfloor = 102$ to a tail pool, and place the remainder in a regular pool. We enumerate all nonempty replica splits and select the one minimizing the sum of the two pools' predicted latencies. The prediction uses an offline per-token latency profile, KV-cache capacity, and batch-size limits, with the 90th and 50th percentiles of the historical length distribution representing the tail and regular pools, respectively. Within each pool, sorted requests are distributed round-robin across replicas and admitted in decreasing predicted-length order on each replica. This allocation can split a prompt group across replicas or across the two pools.

Seer-style Oracle (historical-length request LPT). This baseline uses request-level LPT placement inspired by Seer (Qin et al., 2026). Each response is weighted by its group's historical mean length. Requests are sorted by decreasing weight and greedily assigned to the replica with the lowest accumulated predicted load among those below the equal request-count capacity n/R. Responses from the same prompt group may be placed on different replicas.

The TailSieve executor uses group-sticky placement for the evenly divisible prefix of groups in each pool and request-level scheduling for remainder groups. Group-level selection and completion do not imply that every response of every group is physically assigned to one replica.

The fixed-allocation TailSieve variant uses the same partial-rollout selector and inner cutoff feedback, but holds the replica split at 2:2 for four-TP2-replica configurations and 4:4 for eight-TP1-replica configurations. Full TailSieve additionally adjusts the replica split through its outer loop.

All timings report rollout generation at a settled operating point. The unfinished-group partition initializes the following round, but generation restarts from the original prompt: generated prefixes and cross-round KV state are not reused.

The route-specialization sweep compares symmetric and route-specific proposal depths under the matched depth budget defined in Appendix A.3.

Rollout-concurrency protocol. To isolate the effect of decoding concurrency from speculative decoding, we disable speculation and vary the number of rollout groups under a fixed replica budget. At each concurrency level, the baseline and routing configurations use the same sampled requests, sampling seeds, and generation settings. Table 2 reports the routing-only speedup, where N is the number of prompt groups and each group contains eight responses. For $N = 128$ and $N = 256$, all experimental settings are kept identical to those at $N = 64$, except for the number of rollout groups. See Appendix B.2 for a discussion of the gain regimes.

A.3 Speculative-Decoding Configurations and Depth Selection

With route-specific speculative configurations, changing replica roles can require runtime reconfiguration or additional CUDA-graph captures and persistent buffers. To avoid these switching and memory costs, we evaluate speculative decoding with fixed replica pools while adapting the tail quota.

For equal-sized replica pools, matching the replica-weighted proposal-depth budget reduces to $d_{\text{bulk}} + d_{\text{tail}} = 2d_{\text{base}}$, as used in Figure 6(a). For a general split with R replicas and m tail replicas, the corresponding condition is $(R-m)d_{\text{bulk}} + md_{\text{tail}} = Rd_{\text{base}}$. This budget measures configured proposal depth; actual drafting and verification FLOPs and runtime also depend on the workload and acceptance behavior.

For Figure 6(a), we select route-specific proposal depths through offline profiling on the same 64-prompt workload. At each matched depth budget, we choose the tested pair with the lowest mean generation wall time over settled rounds at the final quota. The selected depths remain fixed at each run.

A.4 End-to-End RL Training Setup

The Qwen3.5-2B and Qwen3.5-4B runs share the rollout settings in Table A.1; Table A.2 lists their batch and training settings. To maintain training stability, we follow Dr. GRPO (Liu et al., 2025b) for advantage normalization: advantages are mean-centered within each eight-response prompt group but are not divided by the within-group reward standard deviation. The RL runs use a separate warm-up batch that is excluded from training. The reward includes a DAPO-style linear overlength penalty (Yu et al., 2025). It is zero up to $8{,}192$ tokens and, for a response of length $L > 8{,}192$, reduces the reward by $(L - 8{,}192)/8{,}192$. This corresponds to a reduction of 0.125 per additional $1{,}024$ tokens and reaches 1.0 at the $16{,}384$-token limit.

The RL runs use the group-first dispatch described in Appendix A.2. The reported timing metric covers rollout generation; optimization and separately timed validation are outside this timer.

Table A.2: Batch and optimizer configuration for the end-to-end mixed-routing RL experiment.

Parameter	Setting
Batch size	64 groups $\times$ 8 responses $=$ 512 trajectories
Training TP	1
Training micro-batch size	8
PPO clip ratio	0.2
Dual-clip coefficient	3.0
KL loss/reward	Disabled
Entropy coefficient	0
Learning rate	2×10^{-7}
Learning-rate schedule	Constant
Adam betas	$(0.9, 0.999)$
Weight decay	0.01
Gradient clipping	1.0
Random seed	42

B Offline Routing Model and Tail Isolation

B.1 Exact Makespan

For a routing policy $p \in \{\text{base}, \text{tail}\}$, let S_r^p denote the set of requests assigned to replica r. The active batch size at decoding position x is

$$b_r^p(x) = \sum_{i \in S_r^p} \mathbf{1}\{L_i \geq x\}.$$

Under the all-admit decoding model, the completion time of replica r is

$$T_r^p = \sum_{x=1}^{L_{r,\max}^p} \tau(b_r^p(x)), \qquad L_{r,\max}^p = \max_{i \in S_r^p} L_i.$$

Equivalently, the same wall-clock time can be distributed evenly among the requests active at each decoding position:

$$T_r^p = \sum_{i \in S_r^p} \sum_{x=1}^{L_i} \frac{\tau(b_r^p(x))}{b_r^p(x)}.$$

To see the equivalence, note that exactly $b_r^p(x)$ requests are active at position x. Therefore,

$$\sum_{i \in S_r^p} \mathbf{1}\{L_i \geq x\} \frac{\tau(b_r^p(x))}{b_r^p(x)} = \tau(b_r^p(x)) \,.$$

Thus, the factor $1/b_r^p(x)$ prevents the same concurrent wall-clock interval from being counted once for every active request.

The exact rollout-step makespan is determined by the slowest replica:

$$T_{\text{step}}^p = \max_r T_r^p,$$

and the exact gain of tail routing is

$$\Delta T_{\text{step}}(q, m) = T_{\text{step}}^{\text{base}} - T_{\text{step}}^{\text{tail}}(q, m).$$

B.2 Tail Workload and Concurrency Effects

Let $\mathcal{T}_q$ denote the isolated tail requests. For request i, let $b_i^p(x)$ denote the active batch size of the replica serving it under policy p; under tail routing, this batch size depends on (q, m). With $[z]_+ = \max\{z, 0\}$, the tail-side proxy decomposes into two competing effects:

$$\widetilde{\Delta T}_{\text{tail}}(q, m) = \underbrace{\sum_{i \in \mathcal{T}_q} \sum_{x=1}^{L_i} \left[\tau\big(b_i^{\text{base}}(x)\big) - \tau\big(b_i^{\text{tail}}(x; q, m)\big)\right]_+}_{\text{low-concurrency advantage}}$$

$$- \underbrace{\sum_{i \in \mathcal{T}_q} \sum_{x=1}^{L_i} \left[\tau\big(b_i^{\text{tail}}(x; q, m)\big) - \tau\big(b_i^{\text{base}}(x)\big)\right]_+}_{\text{tail-concentration penalty}}.$$

This identity holds for any profiled latency curve. The concurrency-based interpretation below applies where $\tau(b)$ is nondecreasing.

Low-concurrency advantage. The first term accumulates the reduction in per-token decoding latency experienced by isolated tail requests. Placing long requests on lightly loaded replicas can lower their decoding latency. This advantage typically arises early, while the baseline replicas still serve many short requests.

Tail-concentration penalty. The second term accumulates the additional per-token latency incurred when isolated tail requests decode more slowly than under the baseline. As short requests finish, baseline concurrency falls, whereas concentrating long requests on only m replicas can sustain higher tail-side concurrency. The resulting penalty can offset the initial advantage. The decomposition separates positive and negative contributions; it does not require a single early-to-late crossover.

The response-length distribution determines both the total isolated work,

$$W_{\text{tail}}(q) = \sum_{i \in \mathcal{T}_q} L_i,$$

and how long requests remain active. The concurrency trajectories $b_i^p(x)$, together with $\tau(b)$, determine the latency difference at each decoding position. Thus, a heavier tail can extend the work that benefits from isolation, but can also prolong the concentration penalty. Increasing q changes the

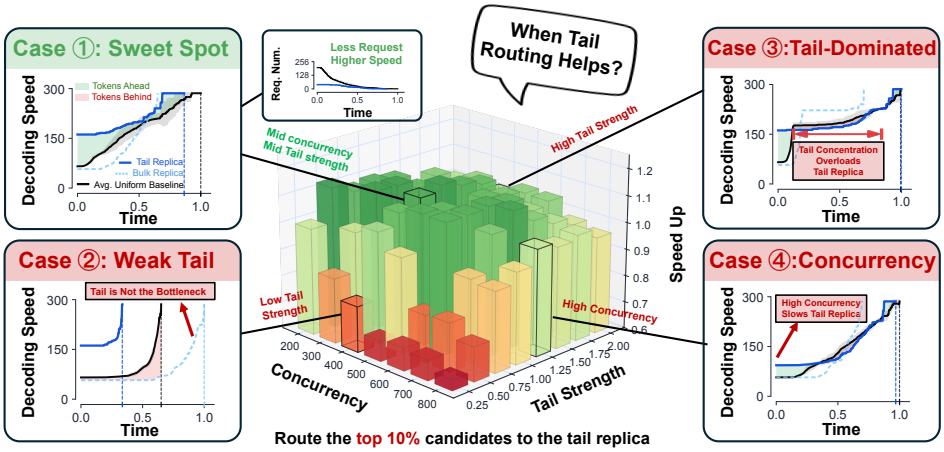


Figure B.1: Gain regimes for fixed top-10% isolation under a fixed replica split.

isolated workload, while changing m at fixed q redistributes that workload across tail replicas. Their best settings therefore depend jointly on tail shape and concurrency-dependent latency, and cannot in general be inferred from tail strength alone.

Unlike the exact per-request decomposition above, this proxy does not divide each shared decoding interval by the number of active requests. It therefore measures aggregate concurrency exposure over the isolated tail set rather than exact wall-clock time. Moreover, the exact rollout-step objective takes the maximum completion time across all replicas, whereas the proxy only describes the isolated tail requests.

We use this proxy to interpret the two effects above. All oracle values and reported speedups are computed using the exact replica-level makespan or measured end-to-end execution time.

Figure B.1 illustrates four representative regimes. Routing gains weaken when the tail workload is too small, the tail route becomes overloaded, or isolation loses its low-concurrency advantage.

B.3 Exact Offline Routing Oracle

This subsection specifies an exact oracle for the two-replica assignment problem used in our motivating analysis. Exactness refers to optimization under a fixed all-admit decoding model; it does not imply an exact model of a serving runtime or an optimal online controller.

Model and scope. There are n requests with known completion lengths $L_1 \geq \cdots \geq L_n \geq 0$. The two replicas are identical. All requests assigned to a replica begin decoding together, advance by one token per decoding iteration, and leave upon completion. The duration of an iteration with b active requests is the fixed, finite value $\tau(b) > 0$ for $1 \leq b \leq n$; set $\tau(0) = 0$. Thus, iteration time depends only on active request count. We exclude prefill, admission queues, memory-capacity constraints, and replica interference from this oracle model. An empty replica has completion time zero. The objective is

$$T^\star = \min_{S \subseteq \{1,\ldots,n\}} \max\{T(S), T(S^c)\}, \tag{6}$$

where T is the replica time in Equation 1.

Sorted insertion cost. Suppose a replica already holds c requests from a prefix of the sorted list, and the next request has length L_i. Each existing length is at least L_i. For the first L_i decoding positions, adding the request changes the active count from c to $c+1$; later positions are unchanged. Consequently, its incremental completion time is

$$d_i(c) = L_i\big[\tau(c+1) - \tau(c)\big]. \tag{7}$$

The formula includes $c = 0$.

State and recurrence. After assigning the first i requests, let $\mathcal{F}_{i,c}$ contain the nondominated attainable completion-time pairs (u, v) for assignments with exactly c requests on replica A and $i - c$ on replica B. A pair (u, v) dominates (u', v') if $u \le u'$ and $v \le v'$, with at least one strict inequality; duplicate pairs need only be stored once. Write ND for removal of dominated pairs and duplicates. Initialize $\mathcal{F}_{0,0} = \{(0, 0)\}$, with all invalid states empty. For $i = 0, \dots, n-1$ and $c = 0, \dots, i+1$, the recurrence is

$$\begin{aligned} \mathcal{C}^{\mathrm{A}}_{i+1,c} &= \{(u + d_{i+1}(c-1), v) : (u, v) \in \mathcal{F}_{i,c-1}\}, \\ \mathcal{C}^{\mathrm{B}}_{i+1,c} &= \{(u, v + d_{i+1}(i-c)) : (u, v) \in \mathcal{F}_{i,c}\}, \\ \mathcal{F}_{i+1,c} &= \mathrm{ND}\big(\mathcal{C}^{\mathrm{A}}_{i+1,c} \cup \mathcal{C}^{\mathrm{B}}_{i+1,c}\big). \end{aligned} \tag{8}$$

The first set is included only when $1 \le c \le i+1$, and the second only when $0 \le c \le i$, so d_{i+1} is never evaluated at a negative count. The oracle returns

$$\min_{0 \le c \le n} \; \min_{(u,v) \in \mathcal{F}_{n,c}} \max\{u, v\}. \tag{9}$$

Storing a predecessor and the replica chosen for each retained pair recovers a corresponding assignment. Different values of c must not be pruned against one another, since their future insertion costs can differ.

Exactness. Without pruning, the two transitions cover every assignment using the exact insertion costs in Equation 7. For candidates in the same state (i, c), any fixed assignment of the remaining requests adds the same pair of future cost increments to each candidate. A dominated pair therefore cannot yield a smaller final makespan than its dominating pair under any continuation. With exact arithmetic and comparisons, pruning preserves an optimal completion, so the terminal minimization in Equation 9 returns $T^\star$.

B.4 Optimality Gaps of Tail Isolation

The reported offline comparison contains 100 sampled assignment instances. Each instance contains 100 trajectories whose lengths are sampled from real rollout traces. The simulator's decoding characteristics are calibrated using direct measurements from Qwen3.5-35B-A3B. For a routing policy with makespan T_{policy}, we measure its relative optimality gap as

$$\frac{T_{\mathrm{policy}} - T_{\mathrm{opt}}}{T_{\mathrm{opt}}},$$

where T_{opt} is the exact offline optimum computed by the dynamic program under the length-known, all-admit decoding model. The reported per-instance best-k isolation policy is within 4% of the model optimum for all 100 instances and within 1% for 72% of them. In comparison, fixed 10% isolation and random uniform routing produce broader, right-shifted gap distributions, as shown in Figure B.2. These simulator aggregates characterize the sampled traces and measured Qwen3.5-35B-A3B serving curve under the length-known simulator.

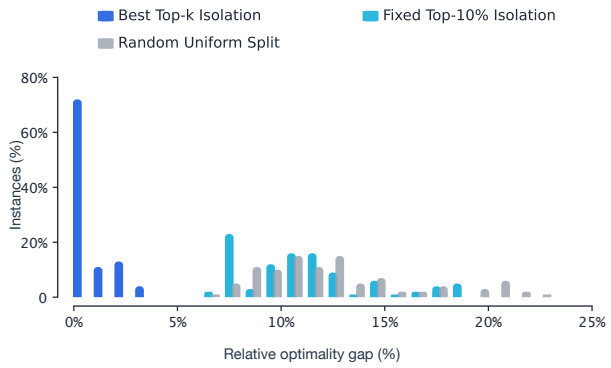


Figure B.2: Model-based optimality-gap distributions across 100 offline assignment instances, each containing 100 trajectories.

B.5 Controlled Tail-Strength Workloads

To vary tail strength without changing request identity or ordering, we derive the weak- and strong-tail workloads from the same measured response-length trace used by the real workload. Let M and P_{90} denote the median and 90th-percentile length of the real trace, respectively. For a target ratio r, we transform only lengths above the median:

$$L_i'(r) = \begin{cases} L_i, & L_i \le M, \\ M + s_r(L_i - M), & L_i > M, \end{cases} \qquad s_r = \frac{rM - M}{P_{90} - M}.$$

The weak- and strong-tail workloads use $r = 1.5$ and $r = 6$, respectively, while the real workload directly uses the original lengths. Because $s_r > 0$, the transformation preserves the ordering of requests. It also leaves the lower half and median unchanged and maps the original P_{90} to rM, making $P_{90}/P_{50} = r$. Thus, the three workloads differ only in the scale of the upper tail. The strong-tail maximum of 22,094 is an artificially transformed simulator length rather than a generated response, and therefore can exceed the 16,384-token cap used in the RL runs. Table B.1 summarizes their resulting length statistics.

Table B.1: Construction and length statistics of the weak-, real-, and strong-tail workloads.

Scene	Length construction	P_{50}	P_{90}	P_{90}/P_{50}	Maximum
Weak Tail	Compress the upper half	1,160	1,740	1.5	3,206
Real Workload	Original lengths	1,160	4,115	3.547	11,800
Strong Tail	Stretch the upper half	1,160	6,960	6.0	22,094

C Validation of the Partial-Rollout Tail Signal

C.1 Partial-Rollout Cutoff and Coverage

Experimental setup. This study uses Qwen3.5-35B-A3B on DeepScaleR with four TP2 vLLM replicas and speculative decoding disabled. Each replica permits at most 256 running sequences. The study follows the mathematics prompt template and sampling settings in Appendix A.1 and Table A.1, with repetition penalty 1.0 and a maximum context length of 16,896 tokens.

Figure 3 uses prompt groups, each containing eight responses. A group finishes when all eight responses complete; its final length is the maximum number of generated tokens among those responses. The cutoff study uses ten fully completed rollout traces, each containing 250 groups, with a completion quota of $M = 200$ groups.

Cutoff definition. For a cutoff parameter ρ, let $k = \rho M$. Within each trace, we take the first $M + k$ groups in submission order. The virtual cutoff is the recorded time at which the M-th group in this subset finishes; the remaining k groups form the candidate set P. This gives

$$\rho = \frac{\text{excess groups}}{\text{completion quota}} = \frac{k}{M}, \qquad M + k = (1 + \rho)M.$$

Thus $\rho = 5\%, 10\%, 25\%$ corresponds to subsets of 210, 220, and 250 groups, leaving 10, 20, and 50 candidates, respectively. The cutoffs are evaluated retrospectively using completion times from the full 250-group traces, so final lengths are available for every candidate.

Coverage and example. Within the same subset, let O be the k groups with the largest final group lengths. Coverage is the fraction of this length-ranked tail set captured by the unfinished candidates,

$$\text{coverage}(\rho) = \frac{1}{10} \sum_{s=1}^{10} \frac{|P_s \cap O_s|}{k}.$$

For example, $\rho = 10\%$ adds 20 groups to the 200-group quota. The virtual cutoff leaves 20 of the 220 groups unfinished; these are compared with the 20 longest groups in that same subset. Across ten traces, 186 of the 200 length-ranked tail groups are selected, giving $186/200 = 93\%$ coverage. The corresponding totals at 5% and 25% are $85/100 = 85\%$ and $474/500 = 94.8\%$. Figure 3(c) rounds these values to whole percentages: 85%, 93%, and 95%.

Cutoff–length mismatch. The cutoff selection and final-length ranking use the same generated responses within each trace. Let $f_{g,r}$ be response r's completion time measured from the start of the rollout, and let $\ell_{g,r}$ be its generated-token count. The two group statistics are

$$F_g = \max_{r=1,\dots,8} f_{g,r}, \qquad L_g = \max_{r=1,\dots,8} \ell_{g,r}.$$

The candidate set P contains the groups with the latest F_g, whereas O contains those with the largest L_g. The serving engine caps the number of running sequences, so responses are admitted at different times as capacity becomes available. Completion times therefore include admission waiting: an earlier-admitted long response can finish before a later-admitted shorter response. Completion order therefore depends on both admission time and generation duration. The yellow uncovered groups in Figure 3(b) represent $O \setminus P$: they belong to the length-ranked tail but have already finished at the virtual cutoff. Thus the mismatch measures the difference between wall-clock completion and token-length rankings within the same rollout.

C.2 Cross-Checkpoint Tail Ranking

We analyze Qwen3.5-2B and Qwen3.5-4B checkpoints to determine whether long-tail behavior is driven primarily by sampling randomness, policy changes, or the prompt. We do not assume that absolute response lengths or the marginal length distribution remain fixed during training. Instead, we ask whether prompt identity remains the dominant source of cross-request length heterogeneity. Let $L_{i,s,r}$ denote the raw response length for prompt i, checkpoint s, and sampling repeat r. We use the same $I = 64$ DeepScaleR prompts, S checkpoints, and $R = 8$ samples per prompt and checkpoint, without normalizing the lengths.

For each prompt and checkpoint, we first average its eight sampled lengths,

$$\bar{L}_{i,s} = \frac{1}{R} \sum_{r=1}^{R} L_{i,s,r}.$$

For a source checkpoint s, we define its tail set $\mathcal{T}_s$ as the $\lceil 0.1I \rceil = 7$ prompts with the largest $\bar{L}_{i,s}$. At a target checkpoint t, we use $\bar{L}_{i,t}$ as the ranking score and compute

$$\mathrm{AUC}(s,t) = \Pr\big(\bar{L}_{i,t} > \bar{L}_{j,t} \mid i \in \mathcal{T}_s,\ j \notin \mathcal{T}_s\big).$$

Figure C.1 reports 0.957–0.985 for adjacent checkpoints and 0.952–0.985 when checkpoint 1 ranks checkpoints through 30, well above the random baseline of 0.5.

C.3 Sources of Response-Length Variation

We further apply a two-factor decomposition directly to $L_{i,s,r}$:

$$SS_{\text{total}} = SS_{\text{prompt}} + SS_{\text{checkpoint}} + SS_{\text{interaction}} + SS_{\text{sampling}}.$$

Using dots to denote averages over the corresponding indices, the main terms are

$$SS_{\text{prompt}} = SR \sum_i (\bar{L}_{i\cdot\cdot} - \bar{L}_{\cdots})^2, \qquad SS_{\text{checkpoint}} = IR \sum_s (\bar{L}_{\cdot s\cdot} - \bar{L}_{\cdots})^2,$$

and

$$SS_{\text{sampling}} = \sum_{i,s,r} (L_{i,s,r} - \bar{L}_{is\cdot})^2,$$

with the interaction given by the remaining between-cell variation. Each contribution is normalized as

$$p_f = \frac{SS_f}{SS_{\text{total}}} \times 100\%.$$

Prompt identity contributes 64.8% of response-length variance for Qwen3.5-2B and 77.78% for Qwen3.5-4B. The corresponding sampling, checkpoint, and prompt–checkpoint interaction contributions are 26.1%, 2.2%, and 6.9% for 2B, and 20.26%, 0.01%, and 1.95% for 4B. Both models show prompt-dominated variation on the evaluated DeepScaleR workload.

These results support the intuition used by TailSieve: long-tail prompts tend to remain long-tailed across adjacent policy updates because differences between prompts dominate the variation . Consequently, TailSieve can use an unfinished prompt group as a noisy cross-round tail signal without requiring the regenerated trajectory to reproduce its previous response length.



Figure C.1: Qwen3.5-2B prompt-tail ranking stability across checkpoints and response-length variance decomposition.

D Controller Implementation and Parameters

TailSieve uses a measured inner loop for the tail-group quota q and a model-based outer loop for the tail-pool replica count m. The inner loop uses equal bulk and tail completion times as its continuous target at fixed m. The outer loop runs only after inner stability is certified by the time-gap tolerance or an adjacent/boundary integer certificate. It compares the current split with its adjacent splits, scoring every split after rebalancing q.

D.1 Measurements and Inner-Loop Stability

Measurements. In round t, T_t^{bulk} is the time until the first $N - q$ of N fresh groups complete, and T_t^{tail} is the completion time of the q tail groups. Only complete, non-transitional rounds at an unchanged quota contribute to rate estimates and stability checks; excluded rounds still count toward elapsed execution time. The normalized time gap is

$$g_t = \frac{T_t^{\text{bulk}} - T_t^{\text{tail}}}{\max\{T_t^{\text{bulk}}, T_t^{\text{tail}}\}}. \tag{10}$$

Stability. The controller uses the EMA rates in Algorithm 1 and treats the inner loop as stable when the absolute mean gap of the latest K eligible observations at the same quota is within ϵ_q. For integer quotas, it may also stop at an adjacent endpoint selected by comparing mean costs when gap signs differ, or at a feasible boundary with a persistent outward-pointing gap.

Fresh observations. After changing q or m, the controller restarts the EMA and collects fresh observations at the new configuration before assessing stability. Transitional or incomplete rounds also restart this measurement window. A new workload identity or a substantial change in workload scale starts a new control epoch with fresh observations.

The 35B, G64 configuration uses the feasible ranges $\mathcal{Q} = \{1, \ldots, 32\}$ and $\mathcal{M} = \{1, 2, 3\}$.

Table D.1: Key controller parameters for the 35B, G64 configuration.

Parameter	Setting
Observation window K	2
EMA decay β	0.7
Work-profile window W	3 complete batches
Balance tolerance ϵ_q	0.05
Replica gain threshold ϵ_m	0.01

D.2 Predicting Completion Times

For each template group i, the work estimate w_i is the median total completion-token count of its accepted observations in the latest W complete batches. The predictor combines these estimates with a fixed, linearly interpolated per-replica aggregate throughput curve $G(c)$ (tokens/s) and cyclic

group assignment. For a replica with sorted group workloads $w_{(1)} \le \cdots \le w_{(h)}$ and $w_{(0)} = 0$, the predicted completion times are

$$F_{(j)} = \sum_{u=1}^{j} \frac{(w_{(u)} - w_{(u-1)})(h - u + 1)}{G(h - u + 1)}. \tag{11}$$

For a candidate split m, the predictor distributes all N fresh groups over $R - m$ bulk replicas; the $(N - q)$th predicted completion gives $\widehat{T}^{\text{bulk}}$. It then reassigns the q groups predicted to finish last to m tail replicas and recomputes their completion times; the latest gives $\widehat{T}^{\text{tail}}$. Enumerating $q \in \mathcal{Q}$ yields the rebalanced score in Eq. (5). The request-profiled throughput curve is evaluated at group concurrency, so these predictions serve as a ranking surrogate rather than calibrated wall-clock estimates.

D.3 Outer Updates and Stopping

Once fresh observations certify inner stability and a current, workload-compatible model snapshot is available, the controller selects the feasible neighbor $m^+ \in \{m-1, m+1\} \cap \mathcal{M}$ with the lowest rebalanced score in Eq. (5). It moves one replica only if

$$\frac{\widehat{\Phi}(m) - \widehat{\Phi}(m^+)}{\widehat{\Phi}(m)} > \epsilon_m. \tag{12}$$

The new quota starts at the predicted minimizer for m^+ and is then rebalanced by the measured inner loop.

The current control episode ends when the inner loop is stable and no neighbor exceeds the gain threshold under the same valid snapshot; without one, the controller holds m and waits. A changed workload or refreshed snapshot can restart adaptation. The fluid-limit analysis in Appendix E.3 supports adjacent search under its assumptions; runtime stopping certifies only observed inner stability and local stationarity of the predictor.

Controller convergence. Figure D.1 shows inner-loop convergence through round-by-round quota updates. Figure 6(b) also shows outer-loop decision convergence to the 2:2 bulk–tail split, with each point reporting rollout time at the corresponding settled configuration.

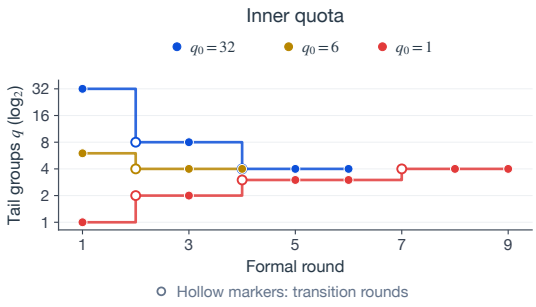


Figure D.1: Inner-quota trajectories for Qwen3.5-35B-A3B with four TP2 replicas and $q_0 = 32, 6, 1$. Hollow markers denote transition rounds.

E Theoretical Properties of Hierarchical Allocation

E.1 Inner-Loop Completion-Time Balance

We first show the balancing property for a two-pool continuous relaxation, where an infinitesimal amount of workload can be shifted between pools and the two completion times are locally differentiable with positive marginal response to the transfer.

Consider two active replicas a and b, with completion times T_a and T_b. Suppose, for contradiction, that an optimal solution satisfies

$$T_a > T_b.$$

Let $\delta = T_a - T_b > 0$. Under the continuous relaxation, we can shift an infinitesimal amount ϵ of workload from the slower replica a to the faster replica b. Let $c_a > 0$ denote the marginal decrease in T_a, and let $c_b > 0$ denote the marginal increase in T_b. For sufficiently small ϵ, the updated completion times satisfy

$$T'_a = T_a - c_a\epsilon + o(\epsilon),$$

and

$$T'_b = T_b + c_b\epsilon + o(\epsilon).$$

Since $T_b < T_a$, we can choose ϵ small enough such that

$$T'_b < T_a.$$

At the same time,

$$T'_a < T_a.$$

Therefore,

$$\max\{T'_a, T'_b\} < T_a = \max\{T_a, T_b\},$$

which contradicts the optimality of the original solution. Thus, under this relaxation, an optimal min-max routing solution equalizes the two active-pool completion times.

In the discrete routing problem, requests are indivisible, so exact equality of replica completion times is not always guaranteed. Instead, the optimal solution is balanced up to the granularity of movable requests. Let S_r denote the set of requests assigned to replica r, and let

$$M = \max_r T_r(S_r)$$

be the makespan of a discrete assignment. For any bottleneck replica a with $T_a(S_a) = M$, if there exists a request $q \in S_a$ and another replica b such that moving q from a to b yields

$$T_a(S_a \setminus \{q\}) < M$$

and

$$T_b(S_b \cup \{q\}) < M,$$

while all other replicas remain below M, then the new assignment has a strictly smaller makespan. This contradicts the optimality of the original assignment. Therefore, a discrete optimal assignment may not make all replicas finish exactly at the same time, but it admits no workload reassignment that can further reduce the maximum completion time.

Discrete rate-model update. TAILSIEVE minimizes the local fixed-rate surrogate over bounded integer quotas in the direction that reduces the observed bottleneck. Its continuous minimizer is Eq. (4); the projected floor and ceiling give its core integer target. The implementation scans admissible integers toward that target and applies a proposal only when it passes the directional, history, and minimum-improvement guards. It then waits for the carry transition to settle before the next update.

The balancing condition also explains the responses observed in the oracle routing result. The oracle does not balance the number of requests across replicas; instead, it assigns a small amount of additional workload to the faster tail replica whenever doing so reduces the bottleneck completion time without making the tail replica the new bottleneck.

E.2 OUTER-LOOP MARGINAL-CAPACITY BALANCE

Continuous condition. We derive the marginal-capacity condition at an interior stationary point of the differentiable two-pool relaxation. This condition interprets the continuous optimum; the runtime controller does not estimate these derivatives or use their ratio to choose a replica direction. Let p be the fraction of routing units assigned to the tail pool and α the fraction of replicas serving it. Define

$$A(p, \alpha) = T^{\text{tail}}(p, \alpha), \qquad B(p, \alpha) = T^{\text{bulk}}(p, \alpha).$$

Locally, moving more traffic to the tail pool makes it slower and the bulk pool faster, whereas moving more replicas to the tail pool has the opposite effect:

$$A_p > 0, \qquad B_p < 0, \qquad A_\alpha < 0, \qquad B_\alpha > 0.$$

For a fixed α, the inner-loop equilibrium $p^*(\alpha)$ satisfies

$$A(p^*(\alpha), \alpha) - B(p^*(\alpha), \alpha) = 0.$$

Implicit differentiation gives

$$\frac{dp^*}{d\alpha} = -\frac{A_\alpha - B_\alpha}{A_p - B_p}.$$

The balanced completion time seen by the outer loop is

$$\Phi(\alpha) = A(p^*(\alpha), \alpha) = B(p^*(\alpha), \alpha)\,.$$

Therefore,

$$\frac{d\Phi}{d\alpha} = A_\alpha + A_p \frac{dp^*}{d\alpha} = A_\alpha - A_p \frac{A_\alpha - B_\alpha}{A_p - B_p} = \frac{A_p B_\alpha - A_\alpha B_p}{A_p - B_p}.$$

At an interior stationary point, $d\Phi/d\alpha = 0$, and hence

$$A_p B_\alpha = A_\alpha B_p, \qquad \text{or equivalently} \qquad \frac{-A_\alpha}{A_p} = \frac{B_\alpha}{-B_p}.$$

The left ratio is the additional tail traffic that can be absorbed per incremental increase in tail capacity without changing the tail completion time. The right ratio is the traffic that must leave the bulk pool under the corresponding capacity loss. Their equality is the marginal-capacity balance condition.

To express the condition as a replica fraction, define the per-replica concurrencies

$$b_{\text{tail}} = \frac{Np}{R\alpha}, \qquad b_{\text{bulk}} = \frac{N(1-p)}{R(1-\alpha)},$$

the local completion-time elasticities

$$\gamma_{\text{tail}} = \frac{\partial \ln A}{\partial \ln b_{\text{tail}}}, \qquad \gamma_{\text{bulk}} = \frac{\partial \ln B}{\partial \ln b_{\text{bulk}}},$$

and the marginal cutoff pressures

$$\lambda_{\text{tail}} = \frac{\partial \ln A}{\partial p}, \qquad \lambda_{\text{bulk}} = -\frac{\partial \ln B}{\partial p}.$$

At the inner-loop equilibrium, where $A = B$, these definitions imply

$$A_p = A\lambda_{\text{tail}}, \qquad B_p = -B\lambda_{\text{bulk}},$$

and

$$A_\alpha = -\frac{A\gamma_{\text{tail}}}{\alpha}, \qquad B_\alpha = \frac{B\gamma_{\text{bulk}}}{1-\alpha}.$$

Substitution into the marginal-capacity condition yields

$$\alpha\lambda_{\text{tail}}\gamma_{\text{bulk}} = (1-\alpha)\gamma_{\text{tail}}\lambda_{\text{bulk}},$$

and therefore

$$\alpha^* = \frac{\gamma_{\text{tail}}\lambda_{\text{bulk}}}{\gamma_{\text{tail}}\lambda_{\text{bulk}} + \gamma_{\text{bulk}}\lambda_{\text{tail}}}.$$

For a homogeneous workload, the two pools have the same local elasticity and differ only through their per-replica routing-unit counts. In that limit, $\lambda_{\text{tail}} \simeq \gamma/p$ and $\lambda_{\text{bulk}} \simeq \gamma/(1-p)$, so the expression reduces to $\alpha^* = p$. The two pools then have equal per-replica loads and are operationally equivalent to uniform routing.

Discrete allocation. The elasticities and cutoff sensitivities are evaluated at the corresponding inner equilibrium. At runtime, the outer controller instead uses the passive finite predictor in Appendix D. It rebalances the quota for the current and feasible adjacent replica splits, and moves at most one replica when the predicted reduction exceeds its threshold.

E.3 Single-Optimum Structure in the Fluid Limit

The outer loop changes the replica split after the inner loop has rebalanced the isolation quota. We analyze the high-concurrency fluid limit and show that its rebalanced outer objective has a single optimum under the conditions below, providing structural support for neighboring-split search.

E.3.1 Fluid Model and Assumptions

We analyze an ideal oracle partition, where the longest requests are assigned to the tail pool before execution and bulk executes only the remaining requests. Let $p = q/N$ be the longest-request fraction assigned to the tail pool, $x = m/R$ its replica fraction, and $\rho = N/R$ the average number of requests per replica. For a response length $L > 0$ almost surely, write $S(z) = \Pr(L > z)$. Let $G(b) > 0$ be the differentiable total decoding throughput of one GPU at concurrency $b > 0$, and let $\tau(b) = b/G(b)$ be the corresponding per-token time. Set $\tau(0) = 0$, the continuous extension implied by the elasticity bounds below. Under ideal within-pool load balancing, the two completion times are

$$T_{\mathrm{tail},\rho}(p, x) = \int_0^\infty \tau\left(\rho \frac{\min\{p, S(z)\}}{x}\right) dz, \tag{13}$$

$$T_{\mathrm{bulk},\rho}(p, x) = \int_0^\infty \tau\left(\rho \frac{(S(z) - p)_+}{1 - x}\right) dz. \tag{14}$$

For fixed x, the tail time increases with p, while the bulk time decreases. Their unique crossing $p_\rho(x)$ minimizes the makespan for that split. The outer loop therefore sees the rebalanced envelope

$$\Phi_\rho(x) = T_{\mathrm{tail},\rho}(p_\rho(x), x) = T_{\mathrm{bulk},\rho}(p_\rho(x), x).$$

Define the throughput elasticity

$$\varepsilon_G(b) = \frac{bG'(b)}{G(b)}.$$

For every $b > 0$, we assume a differentiable, non-increasing elasticity profile bounded away from zero and one:

$$\varepsilon_G'(b) \le 0, \qquad 0 < \underline{\alpha} \le \varepsilon_G(b) \le \overline{\alpha} < 1. \tag{15}$$

Hence $\varepsilon_G(b) \to \alpha_\infty \in (0, 1)$. Let $\gamma = 1 - \alpha_\infty$ and assume

$$\int_0^\infty S(z)^{1-\overline{\alpha}}\, dz < \infty. \tag{16}$$

The bounds apply also at small positive concurrency: at any finite ρ, the completion-time integrals can sample arbitrarily small $\rho S(z)$.

E.3.2 Main Theorem and Proof

Theorem 1 (Unique limiting rebalanced outer optimum)**.** *Suppose that the GPUs are homogeneous, the tail pool receives the longest p fraction of requests, and the inner quota is rebalanced at every replica split. Under Eqs. (15)–(16), as $\rho \to \infty$ the normalized outer objective $\Phi_\rho(x)/\tau(\rho)$ converges uniformly on every compact subinterval of $(0, 1)$ to a limit $\Phi_\infty(x)$. For any non-degenerate Lognormal, Weibull, Gamma, exponential, Uniform, Pareto, or Lomax response-length distribution satisfying the conditions in Table E.1, Φ_∞ strictly decreases before one interior point x^* and strictly increases after it. Thus x^* is the unique global minimizer.*

Proof. Uniform fluid limit. Write $\beta_- = 1 - \overline{\alpha}$ and $\beta_+ = 1 - \underline{\alpha}$, so that $0 < \beta_- \le \gamma \le \beta_+ < 1$. Since $d \log \tau(b)/d \log b = 1 - \varepsilon_G(b)$, integration gives

$$\log \frac{\tau(\rho u)}{\tau(\rho)} = \int_1^u [1 - \varepsilon_G(\rho v)] \frac{dv}{v}.$$

Consequently, for $r_\rho(u) = \tau(\rho u)/\tau(\rho)$,

$$\begin{cases} u^{\beta_+} \le r_\rho(u) \le u^{\beta_-}, & 0 < u \le 1, \\ u^{\beta_-} \le r_\rho(u) \le u^{\beta_+}, & u \ge 1. \end{cases}$$

Taking $\rho = 1$ also proves the continuous extension $\tau(0) = 0$. As $\rho \to \infty$, the logarithmic integral yields $r_\rho(u) \to u^\gamma$, uniformly on every interval $[h, M]$ with $h > 0$. This convergence is uniform on $[0, M]$ as well: for $0 \le u \le h \le 1$, both $r_\rho(u)$ and u^γ are bounded by $h^{\beta-}$.

Fix $K = [\delta, 1-\delta] \subset (0,1)$. For all $p \in [0,1]$ and $x \in K$, each of $u = \min\{p, S(z)\}/x$ and $u = (S(z)-p)_+/(1-x)$ satisfies $0 \le u \le S(z)/\delta$. Thus both the normalized integrand and its limit are bounded by the same envelope:

$$r_\rho(u),\ u^\gamma \ \le\ \max\{u^{\beta-}, u^{\beta+}\} \ \le\ \delta^{-\beta+} S(z)^{\beta-}.$$

Equation (16) makes this envelope integrable. Applying dominated convergence to the supremum of the integrand error over $(p, x) \in [0,1] \times K$ gives uniform convergence of both integrals. This explains the exponent $1 - \overline{\alpha}$ in the assumption: for $u < 1$, the smaller exponent gives the upper bound.

Define

$$a(p) = \int_0^\infty \min\{p, S(z)\}^\gamma dz, \qquad b(p) = \int_0^\infty \left[(S(z)-p)_+\right]^\gamma dz.$$

The two limiting times are $a(p)/x^\gamma$ and $b(p)/(1-x)^\gamma$. Because $L > 0$ almost surely, for every $p < 1$ the set $\{z : S(z) > p\}$ has positive measure. The elasticity bounds make τ strictly increasing, so both the finite-ρ and limiting balance functions are continuous and strictly increasing in p, with opposite signs at $p = 0, 1$. Let $p_\infty(x)$ be the unique root of $F(p, x) = a(p)/x^\gamma - b(p)/(1-x)^\gamma$. Joint continuity and strict monotonicity make p_∞ continuous. For any $\epsilon > 0$, $|F|$ has a positive minimum on the compact set $\{(p, x) \in [0,1] \times K : |p - p_\infty(x)| \ge \epsilon\}$ whenever it is nonempty. The uniform convergence of the balance functions therefore implies $p_\rho \to p_\infty$ uniformly on K. Substituting these roots into the uniformly convergent times proves the stated uniform convergence of $\Phi_\rho/\tau(\rho)$.

Parameterization and derivative. The limiting inner balance has the parameterization

$$x(p) = \frac{a(p)^{1/\gamma}}{a(p)^{1/\gamma} + b(p)^{1/\gamma}}, \qquad \Phi_\infty(x(p)) = D(p)^\gamma, \tag{17}$$

where

$$D(p) = a(p)^{1/\gamma} + b(p)^{1/\gamma}.$$

Because a strictly increases and b strictly decreases, $x(p)$ is a continuous strictly increasing bijection from $[0,1]$ to $[0,1]$. It therefore preserves the order and uniqueness of minimizers.

Let $p = S(t)$ and $k = (1-\gamma)/\gamma$. Differentiating D gives

$$D'(p) = t^{1/\gamma} \left(\left[1 + \frac{e_\gamma(t)}{t} \right]^k - K_\gamma(t) \right), \tag{18}$$

where

$$e_\gamma(t) = \int_t^\infty \left[\frac{S(z)}{S(t)} \right]^\gamma dz.$$

Let $U \sim \mathrm{Uniform}(0,1)$ and $Z_t(u) = S(tu) - S(t)$. The normalized bulk-shape term is

$$K_\gamma(t) = (\mathbb{E} Z_t(U)^\gamma)^{(1-\gamma)/\gamma}\, \mathbb{E} Z_t(U)^{\gamma-1}.$$

For the listed families, the density f is positive and continuous at every interior threshold t. Hence $S(z) - p \sim f(t)(t-z)$ as $z \uparrow t$, so the derivative integral below is finite since $\gamma - 1 > -1$. In particular,

$$a(p) = p^\gamma [t + e_\gamma(t)], \qquad a'(p) = \gamma t p^{\gamma-1}, \qquad b'(p) = -\gamma \int_0^t [S(z) - p]^{\gamma-1} dz,$$

which verifies Eq. (18), including its positive prefactor. Lemma 1 below shows that $e_\gamma(t)/t$ is non-increasing and $K_\gamma(t)$ is non-decreasing. Its family-specific verification gives strict decrease of the first term for Lognormal, Weibull, Gamma, exponential, Lomax, and Uniform laws, and strict increase of the second term for Pareto laws. Consequently, the bracket in Eq. (18) strictly decreases with t and, because $p = S(t)$ decreases with t, $D'(p)$ changes sign at most once, from negative to positive.

Existence of an interior minimizer. Let $M = (\int_0^\infty S(z)^\gamma dz)^{1/\gamma} < \infty$; finiteness follows from $\gamma \geq \beta_-$ and $0 \leq S \leq 1$. For $p \in (0,1)$, set $f_p = \min\{p, S\}$, $g_p = (S-p)_+$, $A = a(p)^{1/\gamma} > 0$, and $B = b(p)^{1/\gamma} > 0$. Then $f_p + g_p = S$. Apply concavity of v^γ to the normalized functions with $\lambda = A/(A+B)$:

$$\frac{\int_0^\infty S(z)^\gamma dz}{(A+B)^\gamma} = \int_0^\infty \left(\lambda \frac{f_p(z)}{A} + (1-\lambda)\frac{g_p(z)}{B}\right)^\gamma dz$$
$$\geq \lambda \int_0^\infty \left(\frac{f_p(z)}{A}\right)^\gamma dz + (1-\lambda)\int_0^\infty \left(\frac{g_p(z)}{B}\right)^\gamma dz = 1.$$

Strict concavity gives equality only if $f_p/A = g_p/B$ almost everywhere. For each listed distribution and every interior p, there is a positive-measure interval beyond t on which $0 < S(z) < p$; there $f_p > 0$ and $g_p = 0$. Since g_p is positive on another interval, the normalized functions cannot agree almost everywhere. Therefore

$$D(p) = A + B < M = D(0) = D(1), \qquad 0 < p < 1.$$

The endpoint values follow directly from the definitions of a, b. Moreover, $f_p^\gamma, g_p^\gamma \leq S^\gamma$, so dominated convergence makes a, b, and hence D, continuous on $[0,1]$. Thus D attains a global minimum, and the strict inequality excludes both endpoints. Its derivative must vanish at an interior minimizer. The strict monotonicity of the bracket in Eq. (18) makes this zero unique, with $D' < 0$ before it and $D' > 0$ after it. Equation (17) transfers these conclusions from p to x. No convexity of D is required. □

E.3.3 Supporting Shape Lemma

Lemma 1 (Monotonicity of the normalized pool shapes)**.** *Let $0 < \gamma < 1$ and let L have a density f supported on an interval $(\ell, r) \subseteq (0, \infty)$, allowing $\ell = 0$ or $r = \infty$. Suppose f is positive and continuous in the interior, $\int_0^\infty S(z)^\gamma dz < \infty$, and the density $g(y) = e^y f(e^y)$ of $\log L$, extended by zero outside its support, is log-concave. For $\ell < t < r$, $e_\gamma(t)/t$ is non-increasing and $K_\gamma(t)$ is finite and non-decreasing. The former is strictly decreasing if $\log g$ is strictly concave on its support. For every family in Table E.1, at least one of these two monotonicities is strict.*

Proof. Regularity and the bulk-shape term. At an interior t, $Z_t(u) \sim tf(t)(1-u)$ as $u \uparrow 1$, while $Z_t(u) \to 1 - S(t) > 0$ as $u \downarrow 0$. Thus both moments defining K_γ are finite, including the negative moment because $\gamma - 1 > -1$. Fix $\ell < t_1 < t_2 < r$, write $Z_i = Z_{t_i}$, and set $q_i(v) = t_i f(t_i v) = g(\log t_i + \log v)/v$. Log-concavity gives the ratio ordering, expressed to include zeros below the lower support endpoint as

$$q_2(v_1)q_1(v_2) \geq q_1(v_1)q_2(v_2), \qquad 0 < v_1 < v_2 < 1.$$

Indeed, on the common positive support the ratio q_2/q_1 is non-increasing, since increments of a concave function decrease with its argument; below that support the inequality follows from the ordered lower support endpoints $\ell/t_2 \leq \ell/t_1$. Since $Z_i(u) = \int_u^1 q_i(v)dv$, integration of this inequality gives

$$\frac{d}{du}\frac{Z_2(u)}{Z_1(u)} = \frac{q_1(u)Z_2(u) - q_2(u)Z_1(u)}{Z_1(u)^2} \leq 0 \quad \text{a.e.}$$

Let $h(u) = \log[Z_2(u)/Z_1(u)]$ and interpolate with $Z_s(u) = Z_1(u)^{1-s}Z_2(u)^s$, $0 \leq s \leq 1$. Both Z_s and h are non-increasing in u. The ratio Z_2/Z_1 has finite positive limits at both endpoints, so h is bounded and the following differentiation is justified by the moment bounds above. Write $\mathbb{E}_{a,s}$ for expectation under the probability density proportional to $Z_s(u)^a$ on $(0,1)$, and set $k = (1-\gamma)/\gamma$. Then

$$\frac{d}{ds}\log\left[(\mathbb{E}Z_s^\gamma)^k \mathbb{E}Z_s^{\gamma-1}\right] = (1-\gamma)(\mathbb{E}_{\gamma,s}h - \mathbb{E}_{\gamma-1,s}h)$$
$$= (1-\gamma)\frac{\mathrm{Cov}_{\gamma-1,s}(Z_s, h)}{\mathbb{E}_{\gamma-1,s}Z_s} \geq 0.$$

The covariance is nonnegative because, for independent U, V under the same tilted measure, twice that covariance equals $\mathbb{E}[(Z_s(U) - Z_s(V))(h(U) - h(V))] \geq 0$. Integration over s proves $K_\gamma(t_2) \geq K_\gamma(t_1)$.

The residual-tail term. Let $H(y) = S(e^y)$ be the survival function of $\log L$. For interior y,

$$\frac{H(y)}{g(y)} = \int_0^\infty \frac{g(y+v)}{g(y)} dv$$

is non-increasing by log-concavity of g. Hence the hazard g/H is non-decreasing and $\log H$ is concave. It follows that $H(y+d)/H(y)$ is non-increasing in y for every $d > 0$, and therefore

$$\frac{e_\gamma(t)}{t} = \int_1^\infty \left[\frac{S(tv)}{S(t)}\right]^\gamma dv$$

is non-increasing. If $\log g$ is strictly concave, the hazard is strictly increasing: for any two interior y values, the integrand in H/g strictly decreases on a positive interval of $v > 0$. The survival ratios then strictly decrease wherever both numerator survival probabilities are positive, which gives strict decrease of the displayed integral.

Verification and strictness for the listed families. Up to additive constants, $\log g(y)$ equals

Lognormal:	$-(y-\mu)^2/(2\sigma^2)$,
Weibull:	$ay - e^{ay}/\lambda^a$,
Gamma:	$ay - e^y/\theta$,
Lomax:	$y - (\kappa+1)\log(1+e^y/\lambda)$.

These functions are strictly concave for the stated positive parameters; exponential is the Gamma case $a = 1$. Thus their residual-tail term is strictly decreasing. For Uniform $[\ell, r]$, $\log g(y) = y + \text{const}$ on its support and

$$\frac{e_\gamma(t)}{t} = \frac{r/t - 1}{\gamma + 1}, \qquad \ell < t < r,$$

which is strictly decreasing, including $\ell = 0$.

For Pareto with lower endpoint $\ell > 0$ and exponent κ, $\log g(y) = \text{const} - \kappa y$ on $y > \log \ell$ is concave. The integrability condition gives $\kappa\gamma > 1$, and $e_\gamma(t)/t = 1/(\kappa\gamma - 1)$ is constant. To prove the needed strictness of K_γ, put $p = (\ell/t)^\kappa$ and $C = p^{-1} - 1$. The exact shape, including the lower-support plateau, is

$$Z_t(u) = pW_C(u), \qquad W_C(u) = \min\{C, u^{-\kappa} - 1\}.$$

The factor p cancels in K_γ, but C strictly increases with t. Set $A(C) = \mathbb{E}W_C^\gamma$, $B(C) = \mathbb{E}W_C^{\gamma-1}$, and $\eta(C) = \Pr(U^{-\kappa} - 1 > C) = (1+C)^{-1/\kappa} > 0$. Differentiating the truncated moments yields $A' = \gamma C^{\gamma-1}\eta$ and $B' = (\gamma-1)C^{\gamma-2}\eta$, so

$$\frac{d}{dC}\log[A(C)^k B(C)] = (1-\gamma)\eta(C)C^{\gamma-2}\frac{CB(C) - A(C)}{A(C)B(C)} > 0.$$

Here $CB - A = \mathbb{E}[W_C^{\gamma-1}(C - W_C)] > 0$, since $0 < W_C < C$ on a set of positive measure. Thus K_γ is strictly increasing for Pareto; scaling does not remove its changing lower-support plateau. □

E.3.4 Covered Distribution Families

Table E.1: Response-length families covered by the single-optimum result.

Distribution	Condition
Lognormal	Non-zero variance
Weibull, Gamma, exponential	Positive distribution parameters
Uniform $[a, b]$	$0 \le a < b$
Pareto	Lower endpoint $\ell > 0$ and $\kappa(1-\overline{\alpha}) > 1$
Lomax	Scale $\lambda > 0$ and $\kappa(1-\overline{\alpha}) > 1$

For Pareto with lower endpoint $\ell > 0$ and Lomax with scale $\lambda > 0$, respectively, direct integration gives, for any $\beta > 0$,

$$\int_0^\infty S(z)^\beta dz = \begin{cases} \ell + \ell/(\kappa\beta - 1), & \text{Pareto,} \\ \lambda/(\kappa\beta - 1), & \text{Lomax,} \end{cases} \qquad \text{if } \kappa\beta > 1,$$

and the integral is infinite otherwise. Thus the table's condition is equivalent to Eq. (16) for these two families. It also implies $\kappa\gamma > 1$, as required by the limiting integrals. The latter condition alone need not provide an integrable envelope for the finite-ρ model.

E.3.5 Implications for Neighboring Replica Search

Corollary 1 (Neighboring replica search in the fluid limit)**.** *Sample the limiting objective on the feasible ordered replica splits $x_j = j/R$. If there is no exact tie, the resulting sequence strictly decreases before its unique minimizer and strictly increases afterward. Starting from any split, repeatedly moving to a lower-valued neighboring split reaches the global minimizer.*

This limiting result motivates the controller's local search. TailSieve evaluates the current and feasible adjacent replica splits, rebalances q for each split in the model, and applies at most one-replica move per decision. The result supports this search only under the stated fluid-limit assumptions; the runtime stopping rule is not a finite-workload optimum proof.

Relation to runtime execution. At runtime, the bulk pool submits N candidate prompt groups, and T^{bulk} is measured when the first $N - q$ groups have completed. This time includes admission waiting and contention from unaccepted candidates before the cutoff. Together with the measured tail completion time, it drives the inner quota loop; the outer replica loop uses the passive predictor in Appendix D.

F Additional Empirical Evaluation

F.1 Routing Gains with Matched Speculative Decoding

Table F.1 separates the combined speedup over non-speculative Uniform routing from the incremental gain over Uniform routing with the same speculative decoder (pure MTP or pure DFlash). All configurations use 64 prompts with eight responses per prompt and eight GPUs: four TP2 replicas for Qwen3.5-35B-A3B and eight TP1 replicas for Qwen3.5-4B and Qwen3.5-2B. Routing fixes equal-sized replica pools and adapts q. Both pools use proposal depth three; DFlash's query block size is four. These comparisons hold proposal depth fixed across routes.

Table F.1: Generation time in seconds and speedup with matched speculative decoding. All models are Qwen3.5. T_0 is Uniform without speculation, T_s is Uniform with speculation, and T_r is TailSieve with the same speculation. Routing gain is T_s/T_r; combined gain is T_0/T_r.

Model	Decoder	T_0	T_s	T_r	Routing gain	Combined gain
DeepScaleR						
35B-A3B	MTP	150.27	72.44	62.87	1.152×	2.390×
4B	MTP	99.94	48.15	41.39	1.163×	2.414×
2B	MTP	62.72	38.57	29.02	1.329×	2.161×
35B-A3B	DFlash	150.27	76.27	68.91	1.107×	2.181×
4B	DFlash	99.94	53.47	47.79	1.119×	2.091×
KodCode-10K						
35B-A3B	MTP	88.16	42.01	34.09	1.232×	2.586×
4B	MTP	71.86	31.43	29.84	1.053×	2.408×
2B	MTP	56.97	30.90	24.26	1.274×	2.349×
35B-A3B	DFlash	88.16	41.41	37.07	1.117×	2.378×
4B	DFlash	71.86	44.65	33.45	1.335×	2.148×

With matched speculative decoding, TailSieve achieves up to 1.33× speedup with MTP and 1.34× with DFlash over uniform routing.

F.2 Downstream Evaluation and Uncertainty

GSM8K and MATH-500 use all 1,319 and 500 test questions, respectively, with greedy decoding (temperature 0, one response per question). Each AIME year contains 30 questions. For every

checkpoint and year, we use three evaluation seeds, with ten responses per question and seed, temperature 0.6, top-$p = 0.95$, and top-k filtering disabled (`top_k=0`). The three seeds use non-overlapping response-sampling streams and are fixed across checkpoints. All evaluations disable thinking, stop at `<END>`, and cap responses at 16,384 tokens. An observed pass@10 score marks a question correct when at least one of its ten responses is correct. Table 4 reports the mean of the three AIME scores; Table F.2 reports the per-seed results and their sample standard deviation.

Table F.2: AIME pass@10 (percent) by evaluation sampling seed; SD is across three seeds, with each checkpoint fixed.

Model	Method	Dataset	Seed 1	Seed 2	Seed 3	Mean ± SD
2B	Base	AIME24	30.00	33.33	36.67	33.33 ± 3.33
2B	Base	AIME25	23.33	26.67	30.00	26.67 ± 3.33
2B	Baseline	AIME24	50.00	63.33	56.67	56.67 ± 6.67
2B	Baseline	AIME25	36.67	36.67	36.67	36.67 ± 0.00
2B	Routing-only	AIME24	53.33	60.00	60.00	57.78 ± 3.85
2B	Routing-only	AIME25	36.67	43.33	43.33	41.11 ± 3.85
2B	Routing + MTP	AIME24	46.67	53.33	60.00	53.33 ± 6.67
2B	Routing + MTP	AIME25	33.33	43.33	33.33	36.67 ± 5.77
4B	Base	AIME24	90.00	93.33	86.67	90.00 ± 3.33
4B	Base	AIME25	83.33	83.33	80.00	82.22 ± 1.92
4B	Baseline	AIME24	90.00	93.33	90.00	91.11 ± 1.92
4B	Baseline	AIME25	83.33	80.00	76.67	80.00 ± 3.33
4B	Routing-only	AIME24	93.33	90.00	90.00	91.11 ± 1.92
4B	Routing-only	AIME25	86.67	80.00	80.00	82.22 ± 3.85
4B	Routing + MTP	AIME24	90.00	90.00	90.00	90.00 ± 0.00
4B	Routing + MTP	AIME25	83.33	86.67	83.33	84.44 ± 1.92

The across-seed standard deviations describe evaluation sampling variation at fixed checkpoints and questions.

F.3 Accepted-Group Length Distributions

Figure F.1 compares Uniform and TailSieve on DeepScaleR across four model configurations. The maximum absolute gap between their empirical CDFs of accepted-group mean response lengths is below 5.3% in every configuration.

G Routing-Induced Length Bias and Quota Transitions

G.1 Setup and Assumptions

We distinguish natural policy-induced distribution drift from bias introduced by routing. The marginal response-length distribution may change as the policy evolves; our question is whether the cross-round routing pipeline shifts the distribution relative to generation under the same current policy and prompt stream. We first analyze an idealized stationary setting with a fixed policy, which isolates the effect of request composition. Let

$$\mathcal{X}_t = \{X_{t,1}, \ldots, X_{t,N}\}$$

be a fresh pool of N requests with prompt distribution P_X. We require consecutive pools to have the same marginal law, $\mathcal{X}_t \overset{d}{=} \mathcal{X}_{t'}$, but do not require them to be independent across steps. Thus, the result also covers, for example, a stationary ordered or without-replacement prompt stream. For a fixed policy π, let $Y(X; \pi, \xi)$ denote the response generated from request X with fresh sampling randomness ξ, and let $L(Y)$ denote its length. For any measurable length interval A, define the prompt-conditional probability

$$p_A(X) = \Pr_{\xi}[L(Y(X; \pi, \xi)) \in A \mid X]$$

and the population response-length distribution

$$P_\pi(A) = \mathbb{E}_{X \sim P_X}[p_A(X)].$$

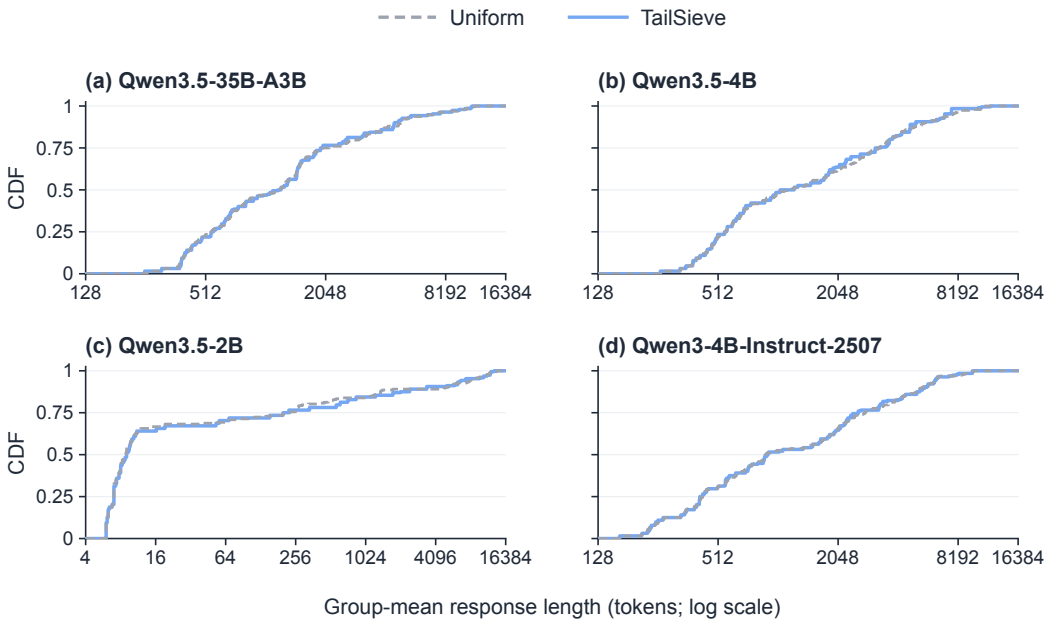


Figure F.1: Empirical CDFs of mean response length per accepted prompt group on DeepScaleR. Each curve pools three evaluation batches of 64 groups, with eight responses per group. Group mean length is the accepted completion-token total divided by eight. Both methods use the same prompt pool and sampling settings; horizontal axes are logarithmic.

For a stable allocation, let $n^{\text{tail}} + n^{\text{bulk}} = N$. We idealize the tail selector as prompt-conditioned: applying the same selection rule to each pool partitions it into

$$\mathcal{X}_t = \mathcal{T}_{n^{\text{tail}}}(\mathcal{X}_t) \uplus \mathcal{F}_{n^{\text{bulk}}}(\mathcal{X}_t),$$

where the two sets contain n^{tail} and n^{bulk} requests, respectively. The selector need not identify the true longest requests; it only needs to depend on the prompt associated with each request and remain unchanged across identically distributed pools. This idealization captures the intuition that long-tail prompts tend to remain long-tailed, while separating that prompt-level signal from trajectory-level sampling noise.

Let $\mathcal{G}_\pi(\mathcal{S})$ denote independent generation under π for every request in $\mathcal{S}$. For a request set $\mathcal{S}$, define its expected number of responses in A as

$$C_A(\mathcal{S}) = \sum_{X \in \mathcal{S}} p_A(X).$$

For a generated update batch $\mathcal{B}_t$, define

$$\widehat{P}_{\mathcal{B}_t}(A) = \frac{1}{N} \sum_{Y \in \mathcal{B}_t} \mathbf{1}\{L(Y) \in A\}.$$

Unless stated otherwise, expectations below are joint over the request pool, fresh generation randomness, and any prompt-conditioned randomness used by the selector. The equality is therefore a population statement: a realized finite batch can deviate from it through sampling variance.

For a fixed-size prompt group selected and consumed as a unit, the same result follows by applying this argument to each response and averaging over the group. This condition concerns selection and update composition; it does not require all responses to execute on one physical replica.

G.2 Stable Tail Allocation

When the two allocations remain fixed, the tail route regenerates the selected requests from the preceding pool, while the bulk route generates the complementary requests from the current pool.

The update batch is therefore

$$\mathcal{B}_t = \mathcal{G}_\pi(\mathcal{T}_{n^{\text{tail}}}(\mathcal{X}_{t-1})) \uplus \mathcal{G}_\pi(\mathcal{F}_{n^{\text{bulk}}}(\mathcal{X}_t)) .$$

Since $\mathcal{X}_{t-1}$ and $\mathcal{X}_t$ are identically distributed and use the same prompt-conditioned selector,

$$\begin{aligned}\mathbb{E}\left[\widehat{P}_{\mathcal{B}_t}(A)\right] &= \frac{1}{N}\mathbb{E}\left[C_A(\mathcal{T}_{n^{\text{tail}}}(\mathcal{X}_{t-1})) + C_A(\mathcal{F}_{n^{\text{bulk}}}(\mathcal{X}_t))\right] \\ &= \frac{1}{N}\mathbb{E}\left[C_A(\mathcal{T}_{n^{\text{tail}}}(\mathcal{X}_t)) + C_A(\mathcal{F}_{n^{\text{bulk}}}(\mathcal{X}_t))\right] \\ &= \frac{1}{N}\mathbb{E}\left[C_A(\mathcal{X}_t)\right] = P_\pi(A).\end{aligned}$$

Thus, repeated generations of the same prompt or prompt group need not have identical lengths. Under the stated fixed-policy idealization, fresh regeneration and cross-round recombination do not introduce routing-induced length bias: $\widehat{P}_{\mathcal{B}_t}(A)$ is an unbiased estimator of the same-policy population $P_\pi(A)$ for every measurable interval A. This comparison is against generation under the same policy; it does not assert that the population distribution remains unchanged across policy updates.

G.3 Increasing the Tail Quota

Suppose the load balancer applies an increase before step t. Write $q = n^{\text{tail}}$ and let $\Delta > 0$, with

$$q^+ = n^{\text{tail},+} = q + \Delta < N, \qquad n^{\text{bulk},+} = N - q^+.$$

The accounting uses the same selection units as above; in group-level execution each unit is a complete prompt group. The incoming queue $\mathcal{C}_t = \mathcal{T}_q(\mathcal{X}_{t-1})$ still contains only q groups. To realize the new quota immediately, the executor admits an additional fresh set $\mathcal{A}_t$, with $|\mathcal{A}_t| = \Delta$, directly to the tail pool. These are new group instances, not retained tail candidates. The bulk pool receives its usual fresh pool $\mathcal{X}_t$ of size N; the extra groups do not enlarge this pool. All tail prompts, including the incoming carry, generate responses under the current policy.

In a successfully completed transition round, all q^+ tail groups and the first $N - q^+$ completed bulk groups enter the update. A group is accepted only after all of its responses have completed. In the prompt-conditioned notation of this analysis, its composition is

$$\begin{aligned}\mathcal{B}_t &= \mathcal{G}_\pi(\mathcal{C}_t) \uplus \mathcal{G}_\pi(\mathcal{A}_t) \uplus \mathcal{G}_\pi\big(\mathcal{F}_{N-q^+}(\mathcal{X}_t)\big) , \\ |\mathcal{B}_t|_{\text{groups}} &= q + \Delta + (N - q^+) = N.\end{aligned}$$

Every incoming carry group is consumed exactly once. The q^+ bulk groups not accepted into this update form the next carry:

$$\mathcal{C}_{t+1} = \mathcal{X}_t \setminus \mathcal{F}_{N-q^+}(\mathcal{X}_t) = \mathcal{T}_{q^+}(\mathcal{X}_t), \qquad |\mathcal{C}_{t+1}| = q^+.$$

Only their prompt identities are retained. Thus the inventory balance is

$$\underbrace{q}_{\text{incoming carry}} + \underbrace{(N+\Delta)}_{\text{fresh admissions}} = \underbrace{N}_{\text{accepted groups}} + \underbrace{q^+}_{\text{outgoing carry}} .$$

For example, when $N = 64$ and q increases from 1 to 2, the transition admits 65 fresh groups: 64 go to bulk and one joins the old carry group in tail. The update accepts $62 + 2 = 64$ groups and leaves two bulk groups as carry. The following round requires only the usual 64 fresh bulk groups.

The quota is therefore active in step t, although its tail pool contains both carried and fresh groups. If the round completes and the allocation remains fixed through step $t+1$, that next step restores the canonical composition, with no fresh tail fill:

$$\mathcal{B}_{t+1} = \mathcal{G}_\pi(\mathcal{T}_{n^{\text{tail},+}}(\mathcal{X}_t)) \uplus \mathcal{G}_\pi(\mathcal{F}_{n^{\text{bulk},+}}(\mathcal{X}_{t+1})) .$$

Under the fixed-policy, identical-pool-law, and prompt-conditioned-selector assumptions stated above, applying the stable-allocation argument gives

$$\boxed{\mathbb{E}\left[\widehat{P}_{\mathcal{B}_{t+1}}(A)\right] = P_\pi(A)}.$$

The transition round t is excluded from settled controller measurements. The next observation becomes eligible only after the unchanged layout completes both quotas with $q_{\text{in}} = q^+$ and no pending transition. Failed or incomplete rounds do not advance this timeline. Restoring the canonical composition establishes the queue accounting; the distributional equality additionally requires the idealized assumptions above and does not follow from counts alone.

G.4 Decreasing the Tail Quota

Let $q = n^{\text{tail}}$ and decrease it by $\Delta > 0$ to $q^- = q - \Delta > 0$ before step t. From the q incoming carry groups $\mathcal{C}_t$, the executor keeps the first q^- in tail, denoted by $\mathcal{H}_t$, and sends the remaining Δ groups $\mathcal{E}_t$ to bulk. Together with $N - \Delta$ fresh groups $\mathcal{U}_t$, they form the mixed bulk pool

$$\mathcal{M}_t = \mathcal{E}_t \uplus \mathcal{U}_t, \qquad |\mathcal{M}_t| = N.$$

The update accepts all of $\mathcal{H}_t$ and the first $N - q^-$ completed bulk groups, denoted by $\mathcal{A}_t$. The next carry is

$$\mathcal{C}_{t+1} = \mathcal{M}_t \setminus \mathcal{A}_t, \qquad |\mathcal{C}_{t+1}| = q^-.$$

It may still contain groups from $\mathcal{E}_t$.

With successful rounds and no further allocation change, step $t + 1$ consumes all of $\mathcal{C}_{t+1}$ in tail, while bulk receives a full fresh pool $\mathcal{X}_{t+1}$. Its outgoing carry therefore comes only from $\mathcal{X}_{t+1}$. The full fresh bulk pool returns at $t + 1$, and the canonical prompt composition returns at $t + 2$:

$$\mathcal{B}_{t+2} = \mathcal{G}_\pi\big(\mathcal{T}_{q^-}(\mathcal{X}_{t+1})\big) \uplus \mathcal{G}_\pi\big(\mathcal{F}_{N-q^-}(\mathcal{X}_{t+2})\big).$$

Under the fixed-policy, identical-pool-law, and prompt-conditioned-selector assumptions above, the stable-allocation argument then gives $\mathbb{E}[\widehat{P}_{\mathcal{B}_{t+2}}(A)] = P_\pi(A)$.

G.5 Evolving Policies and the Deviation Bound

The exact result above fixes the policy and assumes identically distributed prompt pools with an unchanged prompt-conditioned selector. Fresh regeneration samples each response under the current policy, but the learned policy and the partial-rollout selector can both depend on earlier trajectories. We therefore measure the routed batch against the current-policy target P_{π_t}. Write $Y_t = Y(X; \pi_t, \xi_t)$ for the response generated from request X under π_t. Let $Z_t \in \{0, 1\}$ indicate whether request X is selected as a tail candidate at step t, and define

$$s_t(X) = \mathbb{E}[Z_t \mid X], \qquad p_{t,A}(X) = \Pr[L(Y_t) \in A \mid X].$$

Two quantities characterize the deviation from the ideal model:

$$\epsilon_t = \mathbb{E}_X[|s_t(X) - s_{t-1}(X)|]$$

measures cross-policy selector drift, while

$$\eta_t(A) = \mathbb{E}_X[|\mathrm{Cov}(Z_t, \mathbf{1}\{L(Y_t) \in A\} \mid X)|]$$

measures residual trajectory dependence after conditioning on the prompt. For a stable allocation, assuming conditional independence between the regenerated tail response and the preceding selection gives

$$\begin{aligned}\mathbb{E}\Big[\widehat{P}_{\mathcal{B}_t}(A)\Big] - P_{\pi_t}(A) = \mathbb{E}_X\big[p_{t,A}(X)\big(s_{t-1}(X) - s_t(X)\big)\big] \\ - \mathbb{E}_X[\mathrm{Cov}(Z_t, \mathbf{1}\{L(Y_t) \in A\} \mid X)].\end{aligned}$$

The triangle inequality then yields

$$\Big|\mathbb{E}\Big[\widehat{P}_{\mathcal{B}_t}(A)\Big] - P_{\pi_t}(A)\Big| \le \epsilon_t + \eta_t(A).$$

The trace aggregates in Appendix C support using prompt identity as a cross-checkpoint signal, but they do not estimate ϵ_t or $\eta_t(A)$. Thus prompt-dominant length variation supports the selector while the bound keeps its distributional effect explicit.

Across the four examined model configurations on DeepScaleR, TailSieve's mean response length in the accepted batches differs from Uniform by less than 3% in step-wise experiments.

G.6 Summary of Distributional Conditions

The analysis distinguishes three regimes. A stationary policy with identically distributed prompt pools, a stable allocation, and the same prompt-conditioned selector introduces no routing-induced bias in expectation. A stable allocation with policy or selector drift gives the approximation bound

above relative to the current-policy target. An allocation change creates a finite composition transition, during which neither equality is claimed. For every measurable length interval A, the exact fixed-policy special case is

$$\begin{cases} \mathbb{E}[\widehat{P}_{\mathcal{B}_t}(A)] = P_\pi(A), & \text{under stable allocations,} \\ \mathbb{E}[\widehat{P}_{\mathcal{B}_{t+1}}(A)] = P_\pi(A), & \text{one step after increasing the tail allocation,} \\ \mathbb{E}[\widehat{P}_{\mathcal{B}_{t+2}}(A)] = P_\pi(A), & \text{two steps after decreasing the tail allocation.} \end{cases}$$

These equalities compare each routed batch with generation under the same fixed policy; the response-length distribution may still evolve during training. Both timelines start with the allocation change applied before step t, require successful rounds, and assume no further allocation change through $t+1$ for an increase or $t+2$ for a decrease. Under policy or selector drift, the remaining deviation is governed by $\epsilon_t + \eta_t(A)$.

H Detailed Related Work

Overview. The long-tail rollout problem has recently attracted increasing attention. Existing approaches mainly improve rollout efficiency from three perspectives. First, asynchronous or partial-rollout systems (Fu et al., 2025; Zhou et al., 2025; Qu et al., 2025; Kimi Team, 2025) reduce synchronization stalls by decoupling generation from optimization or carrying unfinished trajectories across steps; these designs must manage policy lag or cross-policy trajectory reuse. Second, tail-aware scheduling methods, including RollPacker (Gao et al., 2026) and StreamRL (Zhong et al., 2025), exploit output-length skewness to rebalance generation workloads. RollPacker improves efficiency through workload reorganization, while StreamRL relies on output-length prediction for skewness-aware dispatching. Third, speculative rollout methods (Qin et al., 2026; Shao et al., 2026; He et al., 2026; Xu et al., 2026; Liu et al., 2025a) accelerate decoding through prediction and verification, improving per-request generation efficiency.

H.1 Long-Tail RL Rollouts and Partial Rollout

Long responses create synchronization bubbles in LLM RL. AReaL relaxes this barrier by decoupling rollout generation from training (Fu et al., 2025). Kimi k1.5 reuses segments of previous trajectories (Kimi Team, 2025), while APRIL and CoPRIS overprovision rollouts and carry unfinished trajectories into later steps, with CoPRIS correcting cross-policy reuse through importance sampling (Zhou et al., 2025; Qu et al., 2025). RollPacker instead consolidates tail prompts into a small number of long rounds; its short rounds also generate extra responses per prompt and retain the first ones to finish (Gao et al., 2026).

TailSieve uses unfinished requests differently. Their cutoff-time status serves only as a training-free tail signal: generated prefixes are not reused, and every consumed response is regenerated from the original prompt under the current policy. All responses in each accepted prompt group are retained. TailSieve mixes regenerated tail groups with fresh bulk groups at each update and targets replica-level makespan within synchronous rollout.

H.2 Length-Aware Scheduling and Routing

Serving schedulers use proxy models, uncertainty-aware distributions, or entropy-guided representations to predict response lengths (Qiu et al., 2024; Zheng et al., 2026; Xie et al., 2026). In RL systems, StreamRL uses a learned output-length ranker for skewness-aware dispatching (Zhong et al., 2025), while Seer exploits similarities among responses to the same prompt through online context learning and divided rollout (Qin et al., 2026).

TailSieve does not estimate exact completion lengths. It uses partial rollout as a noisy prompt-level tail filter and selects tail workload at group granularity. A fast inner loop adjusts the isolation quota q from measured pool barriers, while a slower outer loop adjusts replica capacity through calibrated local predictions. The tail filter needs no auxiliary length-prediction model.

H.3 SPECULATIVE ROLLOUT DECODING

Speculative rollout methods reduce generation cost using adaptive drafts, history from nearby rollouts, idle-compute pre-generation, or improved multi-token prediction (Shao et al., 2026; He et al., 2026; Liu et al., 2025a; Xu et al., 2026; Li et al., 2026). BubbleSpec uses idle rollout capacity to pre-generate drafts, while DAS adapts speculation budgets using historical lengths; both employ suffix-based drafting (Xu et al., 2026; Shao et al., 2026). These techniques accelerate token generation, whereas TAILSIEVE shapes the concurrency at which generation runs. The two dimensions are complementary: tail isolation creates a low-concurrency, long-horizon route on which deeper MTP or DFlash speculation can be applied without imposing the same policy on the high-concurrency bulk route. Section 3.2 reports the outcome of the suffix-decoding configurations explored in our implementation.