RANA: ACCELERATING MULTI-TURN AGENTIC REINFORCEMENT LEARNING WITH SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

The evolution of Large Language Models (LLMs) toward reasoning-intensive agents relies heavily on Reinforcement Learning (RL) for alignment, e.g., RL human feedback (RLHF). However, the RLHF rollout phase suffers from severe computational inefficiencies, particularly in multi-turn interactions where sequential dependencies exacerbate the long-tail effect and lead to GPU underutilization. While recent work has explored speculative decoding (SD) to accelerate RLHF rollouts, they overlook the structural properties of agentic multi-turn interactions, i.e., contextual continuity, entropy decay, and hardware sensitivity, leaving substantial optimization potential unexploited. In this paper, we propose RANA (**R**ollout **A**daptive **N**-Gram **A**ccelerator), a system that accelerates multi-turn agentic RLHF rollouts by systematically exploiting these structural properties. RANA consists of three innovative components: (1) a tree-based N-gram speculative decoding mechanism that leverages historical context for model-free drafting; (2) a load-adaptive controller that dynamically toggles speculation based on Roofline analysis to avoid compute-bound contention; and (3) an entropy-aware scheduler that leverages entropy decay to align low-entropy rollouts with the execution tail, where speculative decoding yields the greatest benefit. Experiments show that RANA achieves up to $3.74\times$ speedup, significantly outperforming prior approaches.

1 INTRODUCTION

The landscape of Large Language Models (LLMs) has been fundamentally reshaped by the emergence of reasoning-intensive architectures, exemplified by OpenAI-o1 (Jaech et al., 2024) and Deepseek-R1 (Guo et al., 2025). These models leverage Reinforcement Learning (RL) to facilitate extended reasoning processes, driving substantial breakthroughs in complex reasoning domains such as mathematics and coding (Wang et al., 2025b; Zhou et al., 2025). Building upon these reasoning capabilities, the research frontier is shifting toward autonomous agents required to formulate sequential plans and interact dynamically with external environments (Li et al., 2025; Wang & Ammanabrolu, 2025). To bridge the gap between static reasoning and such agentic tasks, which demand a robust understanding of environmental dynamics and state transitions, RL remains the de facto alignment mechanism (Ouyang et al., 2022). However, as these workflows evolve into increasingly complex, multi-turn interactions, the computational inefficiency of the underlying RL pipeline becomes a critical impediment (Zhang et al., 2025).

RLHF training pipelines are severely bottlenecked by long-tail effect during the rollout phase. In standard practice, prompts are processed in batches; while the inference engine initially operates at saturation, the completion of shorter requests leads to a diminishing batch size. This leaves the engine underutilized, processing only a few straggling requests while blocking the pipeline. While prior research has mitigated this issue in single-turn scenarios, multi-turn interactions introduce a structurally more complex challenge. Unlike independent single-turn requests, multi-turn sessions impose strict sequential dependencies, which inherently prevent the parallel execution of consecutive turns within a single request. Consequently, a session concludes only after its final turn, causing computational delays to compound across turns. This dependency drastically exacerbates the long-tail effect, resulting in severe GPU underutilization and prohibitive training overhead.

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.

Crucially, we argue that this multi-turn paradigm, while intensifying latency, simultaneously reveals unique structural properties that can be harnessed for optimization. We identify three key phenomena: (1) *Contextual Continuity.* Multi-turn interactions typically involve the iterative refinement of a single, invariant problem defined by the initial prompt. Consequently, the solution space remains highly constrained throughout the session, rather than expanding arbitrarily. This constraint induces strong semantic and structural coherence across turns: the model's Chain-of-Thought (CoT) in the current turn (T_i) frequently shares substantial lexical and logical overlap with the preceding turn (T_{i-1}), as the model tends to restate premises or modify previous reasoning paths within the same limited context. (2) *Hardware Regime Sensitivity.* The efficacy of Speculative Decoding (SD) is intrinsically strictly coupled with the inference engine's operational regime. While SD effectively utilizes spare bandwidth in memory-bound states (low load), the additional arithmetic overhead for verification introduces critical resource contention in compute-bound states (high load). In high-throughput RLHF rollouts, where batch sizes frequently saturate GPU streaming multiprocessors, this contention often outweighs the speculative gains, leading to *negative acceleration*. Consequently, a blind, always-on speculation strategy is detrimental. (3) *Entropy Decay.* Output entropy decays with turn depth, making later turns significantly more amenable to Speculative Decoding (SD). Since SD is typically activated only during the low-load tail to avoid resource contention, the composition of this tail is critical. This indicates that the potential for speculative acceleration is non-uniformly distributed across tasks. Consequently, the maximum reduction in tail latency is theoretically achievable only when the bottleneck phase is aligned with these high-acceptance, low-entropy tasks, rather than being bounded by high-entropy outliers.

Motivated by these insights, we propose **R**ollout **A**daptive **N**-Gram **A**ccelerator (RANA), a holistic framework designed to synchronize speculative execution with both the semantic structure of the task and the physical state of the hardware. RANA directly addresses the identified multi-turn properties through three integrated mechanisms. First, to exploit *Contextual Continuity*, we implement a **Trie-based N-gram Drafter** that harvests historical redundancy for model-free speculation. Second, to navigate *Hardware Regime Sensitivity*, we introduce a **Load-Adaptive Controller** that dynamically toggles speculation based on real-time compute saturation, ensuring decoding strategies align with the hardware Roofline. Finally, to leverage *Entropy Decay*, we deploy an **Entropy-Aware Scheduler** that strategically defers low-entropy, convergent turns to the memory-bound tail phase, maximizing the effective acceptance rate where it matters most.

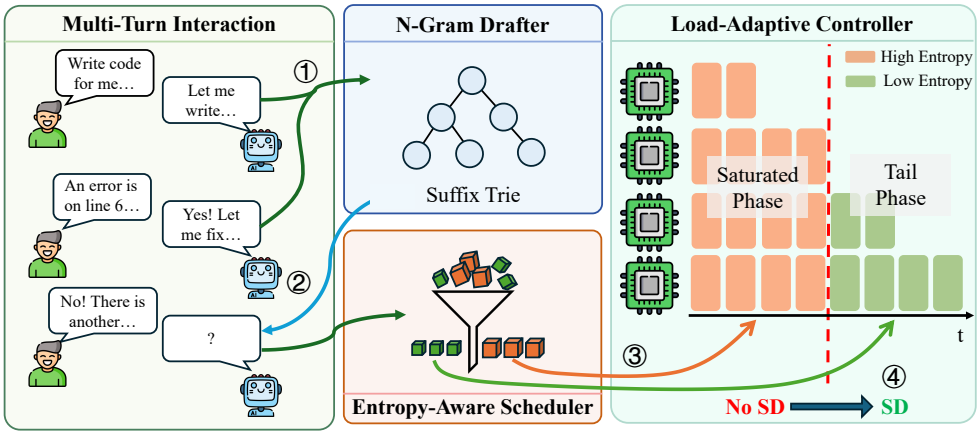


Figure 1: System architecture of RANA. The N-gram drafter ① continuously updates its corpus with historical actions. Upon new request arrival, the system performs ② draft-then-verify inference. The entropy-aware scheduler ③ orchestrates execution by dispatching high-uncertainty tasks first, regulated by the load-adaptive controller, which ④ activates speculation exclusively during memory-bound (profitable) regimes.

2 BACKGROUND AND RELATED WORK

2.1 EFFICIENT SCHEDULING FOR RLHF ROLLOUT

Prior work has largely focused on accelerating the rollout phase in non-agent, single-turn scenarios. **RLHFuse** (Zhong et al., 2025b) integrates rollout and training via a collocation strategy, dynamically repurposing workers to minimize GPU bubbles. Similarly, **StreamRL** (Zhong et al., 2025a) decouples generation and training modules using streaming transmission to mitigate batch fragmentation. To address the straggler problem, **Rollpacker** (Gao et al., 2025) employs tail batching to reschedule long-tail requests, improving GPU utilization while maintaining on-policy constraints. While effective for single-turn dialogue, these approaches overlook the structural properties inherent to multi-turn agentic interactions—such as context dependency and varying computational loads across turns—leaving significant optimization potential unexploited.

2.2 ROLLOUT IN MULTI-TURN RLHF

In the RLHF rollout phase, full interaction trajectories are sampled from the current policy and subsequently scored by a reward model (Zhang et al., 2025). We formalize this process for multi-turn tasks as follows.

Consider a batch of N independent tasks $\mathcal{B} = \{S_1, S_2, \ldots, S_N\}$. Each task S_i represents a multi-turn session initiated by a prompt $p_{i,0}$. The state of task S_i at turn t is denoted by $s_{i,t}$, with $s_{i,0} = \{p_{i,0}\}$. To manage execution, the inference engine maintains three queues: the waiting queue $\mathcal{Q}_{\text{wait}}$, the active queue $\mathcal{Q}_{\text{run}}$, and the completion queue $\mathcal{Q}_{\text{done}}$. Initially, all tasks reside in $\mathcal{Q}_{\text{wait}}$.

The system operates iteratively. At each scheduling step, the engine dispatches tasks from $\mathcal{Q}_{\text{wait}}$ to fill available GPU slots in $\mathcal{Q}_{\text{run}}$. For an active task S_i at turn t, the policy generates an action $a_{i,t}$ conditioned on the current state $s_{i,t}$. Upon generation completion, the task releases its slot in $\mathcal{Q}_{\text{run}}$, and the state updates by appending the action and the subsequent environment instruction:

$$s_{i,t+1} \leftarrow s_{i,t} \oplus a_{i,t} \oplus p_{i,t+1}. \tag{1}$$

Here, $\oplus$ denotes concatenation, and $p_{i,t+1} = \mathcal{E}(s_{i,t}, a_{i,t})$ represents the environment transition from ground truth or simulated user feedback. It then returns to $\mathcal{Q}_{\text{wait}}$ for the next turn or moves to $\mathcal{Q}_{\text{done}}$ if a termination condition is met.

2.3 SPECULATIVE DECODING IN RLHF ROLLOUT

In this section, we first present a formal formulation of Speculative Decoding to establish the theoretical groundwork. Subsequently, we survey prior works that leverage speculative techniques to accelerate RLHF rollouts, highlighting the context for our proposed approach.

Standard Large Language Model (LLM) inference is memory-bandwidth bound due to its serial autoregressive nature, where generating a token x_t requires loading all model parameters. Speculative Decoding (SD) mitigates this via a *Draft-then-Verify* paradigm (Chen et al., 2023; Leviathan et al., 2023).

Formally, given a context $x_{<t}$, a lightweight drafter $\mathcal{D}$—typically a small auxiliary model or heuristic—rapidly hypothesizes a candidate sequence $\tilde{x}$ of length γ. These speculative tokens are verified in parallel by the target model $\mathcal{M}$, which computes the probability distributions for the sequence $\{x_{<t} \oplus \tilde{x}_{1:i}\}$ for all $i \leq \gamma$ in a single forward pass. The system accepts the longest prefix consistent with the target model's distribution. The acceptance length k is defined as:

$$k = \max\{i \mid \tilde{x}_j = \mathcal{M}(x_{<t} \oplus \tilde{x}_{<j}), \forall j \leq i\}. \tag{2}$$

This mechanism allows the engine to advance by $k + 1$ tokens per invocation of $\mathcal{M}$, significantly reducing memory access overhead (Fu et al., 2024).

Speculative decoding in RLHF. Recent efforts to accelerate RLHF via Speculative Decoding generally fall into two categories: auxiliary model-based and retrieval-based approaches. Methods like **ReSpec** (Chen et al., 2025) and **SpecActor** (Cheng et al., 2025) employ synchronized small models or decoupled drafters to guide exploration; however, they incur significant compute and

memory overheads due to model maintenance. Conversely, statistical approaches such as **Seer** (Qin et al., 2025), **RhymeRL** (He et al., 2025), **RLHFSpec** (Wang et al., 2025a), **SPEC-RL** (Liu et al., 2025), and **SRT** (Chang et al., 2025) leverage historical data or idle cycles to bypass model execution. While lightweight, these retrieval methods suffer from the "cold start" problem due to their reliance on cross-epoch history and are primarily optimized for single-turn scenarios, failing to exploit unique characteristic of multi-turn agentic tasks.

3 METHOD

As illustrated in Figure 1, RANA accelerates multi-turn rollout via three synergistic components: an **Efficient N-gram Drafter** (§3.1) that exploits temporal locality for model-free drafting; a **Load-Adaptive Controller** (§3.2) that toggles speculation according to real-time compute saturation; and an **Entropy-Aware Scheduler** (§3.3) that defers low-entropy tasks to the memory-bound tail where speculation is most profitable.

3.1 EFFICIENT N-GRAM DRAFTER

Centered on an enhanced N-Gram Trie, the **Efficient N-Gram Drafter** employs an optimized retrieval algorithm with proven **amortized** $O(1)$ **complexity**. To accommodate diverse RL paradigms, it features a flexible *Session Management* mechanism that enforces isolation for independent sessions, while enabling *Inter-Group Aggregation* for ensemble methods like GRPO to share context indices.

Generating draft from history. Diverging from parametric approaches, RANA employs a dynamic, non-parametric drafter $\mathcal{D}$ grounded strictly in session history. For a task at turn t, $\mathcal{D}$ maintains a reference corpus $\mathcal{C} = \{a_0, \ldots, a_{t-1}\}$ containing prior actions. To synthesize a draft $\tilde{x}$, let s_{j-1} denote the suffix of length K from the current context. The speculative token $\tilde{x}_j$ is determined by maximizing its empirical frequency within $\mathcal{C}$:

$$\tilde{x}_j = \arg\max_{v \in \mathcal{V}} \sum_{a \in \mathcal{C}} \text{Count}(s_{j-1} \oplus v, a). \tag{3}$$

Iterative application of Eq. (3) yields a candidate sequence that effectively retrieves recurring patterns from past reasoning chains. This draft is subsequently submitted to the target model $\mathcal{M}$ for parallel verification, converting historical redundancy into inference acceleration.

Dynamic lookup via suffix trie with failure links. The core computational challenge in Eq. (3) lies in efficiently querying the reference corpus $\mathcal{C}$ for arbitrary suffixes. While hash-based implementations allow for $O(1)$ retrieval of fixed-length N-grams, they lack the flexibility to handle variable-length contexts required for dynamic longest-match queries. To address this, we structure the reference corpus $\mathcal{C}$ as a *Bounded Suffix Trie* $\mathcal{T}$ with a maximum depth of $L + 1$, where L represents the maximum context lookahead window. Each node u in $\mathcal{T}$ represents a specific token sequence (substring) observed in the history. To support rapid context switching, we augment the structure with Aho-Corasick-style failure links (Aho & Corasick, 1975). Specifically, for a node u representing a string S, we maintain a failure pointer $\pi(u)$ directed to the node representing $S[1:]$ (the longest proper suffix of S). Each node further maintains a counter $C(u)$ and a set of outgoing edges to children nodes, enabling the direct computation of token frequencies required by Eq. (3).

The inference matching process operates as a deterministic state machine, with the full pseudocode summarized in Algorithm 1 (Appendix A). When extending the draft, the traversal follows edges for matching tokens and transitions via failure pointers $\pi(u)$ upon mismatches. This fallback mechanism guarantees finding the longest matching suffix in the history without restarting the search. For prediction, the algorithm selects the child node v maximizing $C(\text{child}_v)$ at the matched state.

Amortized $O(1)$ retrieval via state persistence. Optimizing traversal complexity is critical for high-throughput inference. RANA combines an *intra-step sliding window*—which uses failure pointers to shift the lookahead by one position when a full depth-$(L{+}1)$ match is reached, reducing per-step cost from $O(L \times \gamma)$ to $O(L + \gamma)$—with inter-step *State Persistence*, which carries the trie pointer u_t across decoding steps so that the starting state u_{t+1} is reached for free along the

verified prefix. By the aggregate property of the Aho-Corasick automaton, the total number of state transitions over a session of length T is bounded by $2T$, yielding amortized retrieval cost $\mathcal{C}_{\text{amortized}} \leq 2T/T = O(1)$ per token, independent of the history window L or the speculative depth γ. The full derivation is given in Appendix B.1.

Online corpus update via anticipatory node instantiation. To incorporate verified tokens online, we insert all length-$(L{+}1)$ substrings of an update buffer $B = x_{\text{suffix}} \oplus x_{\text{acc}}$ into $\mathcal{T}$. To avoid expensive failure-link backtracking, we use *Anticipatory Node Instantiation*: when creating a node u for string S, we set $\pi(u)$ to the node for $S[1{:}]$, eagerly materializing it as a zero-counter placeholder if absent. Since $S[1{:}]$ is the prefix of the next substring in B, every anticipatory node is visited and updated at the next insertion step, yielding strictly linear-time updates.

Homologous corpus densification via hierarchical scope. To balance corpus density with semantic relevance, RANA manages tries hierarchically: *Session Isolation* maintains a distinct trie $\mathcal{T}_i$ per task to suppress cross-task hallucinations; *Shared Session Scope* pools all G trajectories of a single prompt under one trie for GRPO/Best-of-N ensembles; and *Cross-Epoch Persistence* retains tries across training epochs, exploiting the bounded policy drift enforced by RLHF's KL constraint to reuse high-correlation patterns between iterations E and $E{+}1$.

3.2 Load-adaptive controller

While the N-Gram Drafter (§3.1) significantly reduces latency by leveraging memory-bandwidth slack, its practical deployment faces a critical system-level challenge: *resource contention* under high throughput.

The verification of γ speculative tokens introduces a arithmetic overhead per request. In low-concurrency scenarios, this overhead is effectively masked by the memory-bound nature of LLM inference. However, as the batch size increases, the GPU's streaming multiprocessors (SMs) become saturated, transitioning the system to a compute-bound regime. In this state, the additional FLOPS required for speculative verification contend with the target model's forward pass for finite compute resources. This contention can offset the acceleration benefits of speculative decoding, potentially resulting in a net performance loss under high throughput.

To ensure robust performance across diverse traffic patterns, RANA incorporates a *Load-Adaptive Controller*. We define a saturation threshold, τ_{sat}, derived from the hardware's roofline performance. At each scheduling step t, the regulator monitors the active batch size B_t of the execution queue. The speculative depth γ is then dynamically modulated via a thresholding policy accordingly:

$$\gamma_t = \begin{cases} \gamma_{\text{default}} & \text{if } B_t \leq \tau_{\text{sat}}, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

By forcing $\gamma = 0$ when B_t exceeds τ_{sat}, RANA gracefully falls back to standard decoding during throughput-critical peaks. This mechanism guarantees that the system opportunistically exploits speculative acceleration in latency-sensitive phases while protecting throughput stability during saturation, effectively making inference efficiency robust to workload fluctuations.

Hyperparameter calibration. While the saturation threshold τ_{sat} is inherently related to the hardware configuration, in practice, its optimal value is influenced by dynamic factors such as the evolving complexity of the policy model and the increasing average context length across training epochs. As the rollout distribution shifts over time, a fixed threshold may drift from optimality. To address this, RANA implements a *Hyperparameter Calibration* mechanism.

At the onset of each RLHF training epoch, prior to full-scale rollout, the system executes a lightweight profiling phase using a mini-batch of calibration $\mathcal{B}_{\text{cal}}$. We perform a rapid grid search over a candidate set of thresholds Λ. The system empirically measures the rollout makespan T_{rollout} for each candidate configuration, and the parameter λ^* that yields the minimum latency is selected as the operational hyperparameter for the current epoch:

$$\lambda^* = \arg\min_{\lambda \in \Lambda} T_{\text{rollout}}(\mathcal{B}_{\text{cal}} \mid \lambda). \tag{5}$$

This calibration process incurs negligible overhead while ensuring that the Load-Adaptive Controller remains synchronized with the instantaneous throughput capabilities of the infrastructure. A formal Roofline-model derivation that grounds the existence of τ_{sat} and characterizes the resulting speedup regimes is provided in Appendix B.2.

3.3 Entropy-aware scheduler

While the Load-Adaptive Controller (§3.2) regulates speculation depth, it remains agnostic to task difficulty, typically defaulting to First-In-First-Out (FIFO) dispatching. This is suboptimal for heterogeneous multi-turn workloads: FIFO risks processing highly predictable tasks during compute-saturated phases (where speculation is disabled), leaving complex, low-acceptance tasks for the memory-bound tail. To maximize acceleration, RANA's **Entropy-Aware Scheduler** reorders dispatching from $\mathcal{Q}_{\text{wait}}$ to ensure that predictable tasks are reserved for the tail phase where speculation is most effective.

Entropy decay. Our strategy leverages the observation that intrinsic uncertainty evolves during iterative reasoning. Let $H(S_{i,t})$ denote the average token-level Shannon entropy. We identify two key properties: (1) **Temporal Decline:** $H(S_{i,t})$ typically decreases as turn depth t increases, reflecting a transition from high-divergence exploration to low-divergence convergence, as empirically confirmed in Figure 2. (2) **Inverse Correlation:** Entropy is inversely proportional to the N-Gram acceptance rate α. Consequently, low-entropy, later-turn interactions represent the ideal workload for speculation and are the target resource for our scheduler.

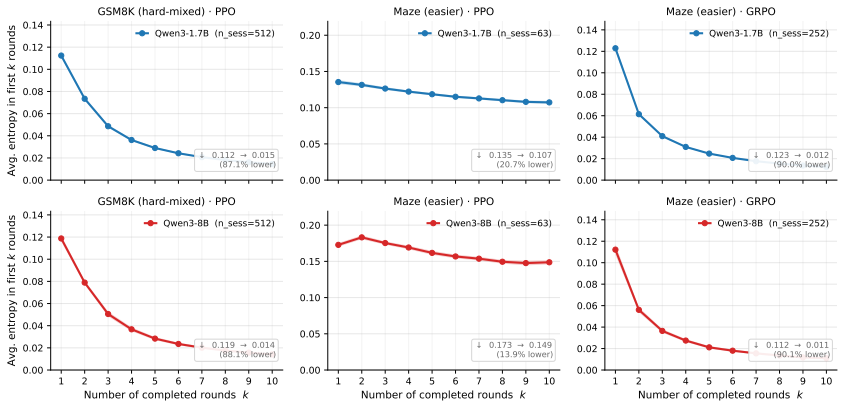


Figure 2: Average per-round token-level entropy decreases monotonically with more interaction rounds, empirically validating the *Temporal Decline* property that motivates RANA's entropy-aware scheduler.

Stratified scheduling. Building on the Load-Adaptive Controller, the rollout lifecycle is bifurcated into two distinct operational regimes governed by the saturation watermark τ_{sat}. During the **Saturated Phase** ($B_t > \tau_{\text{sat}}$), the system operates in a compute-bound state where speculation is intentionally disabled ($\gamma = 0$) to prioritize aggregate throughput. Conversely, as the active batch size diminishes, the system enters the **Tail Phase** ($B_t \leq \tau_{\text{sat}}$); here, inference transitions to a memory-bound regime, triggering the activation of speculative decoding ($\gamma > 0$) to minimize latency for the remaining tasks.

To optimize global efficiency, the scheduler employs a *Stratified Scheduling Strategy*. The core intuition is to process "hard" (high-uncertainty) tasks during the Saturated Phase where speculation is anyway disabled, while reserving "easy" (low-uncertainty) tasks for the Tail Phase to maximize the speedup from enabled speculation.

Formally, for each task S_i residing in the waiting queue $\mathcal{Q}_{\text{wait}}$, we utilize the average entropy of the preceding turn, $H(S_{i,t-1})$, as a proxy metric for the expected uncertainty of the current turn. When the execution queue $\mathcal{Q}_{\text{run}}$ has available slots, the scheduler prioritizes tasks according to a descending entropy order:

$$S_{\text{next}} = \underset{S_i \in \mathcal{Q}_{\text{wait}}}{\arg\max}\, H(S_{i,t-1}). \tag{6}$$

By greedily dispatching high-entropy tasks, the system effectively flushes out the high-uncertainty workload while the engine is saturated. Consequently, as the queue drains and the system enters the Tail Phase, the remaining tasks in $\mathcal{Q}_{\text{wait}}$ are predominantly low-entropy, convergent interactions. Since the acceptance rate α is inversely correlated with entropy, this descending-entropy dispatch provably minimizes the aggregate tail entropy and thus yields a strictly higher effective acceptance rate than FCFS, i.e., $\mathbb{E}[\alpha_{\text{RANA}}] > \mathbb{E}[\alpha_{\text{FCFS}}]$; the formal statement is given in Appendix B.3.

4 EVALUATION

To evaluate the efficacy of RANA, we conducted experiments using diverse base models across a range of Agentic multi-turn tasks. Our results are summarized below.

4.1 EXPERIMENTAL SETUP

Implementation. RANA is developed as a lightweight, high-performance extension to the **verl** framework (v0.6.1) (Sheng et al., 2025), integrating a modified **SGLang** (v0.5.4.post3) (Zheng et al., 2024) inference kernel to support dynamic suffix-trie management. The system is architected as a transparent, drop-in replacement for the standard multi-turn rollout backend. This design ensures full compatibility with existing reward modeling and policy optimization pipelines, allowing RANA to accelerate training without requiring modifications to the core RL algorithms or model architectures.

Benchmarks. To rigorously evaluate RANA, we curate three diverse agentic benchmarks. For symbolic self-correction, we adapt **GSM8K** (Cobbe et al., 2021) via the **MINT** (Wang et al., 2024) framework; this multi-turn refinement format specifically tests the drafter's ability to leverage immediate reasoning templates. For spatial state-tracking, we employ **Maze** (Shu et al., 2025), a partially observable navigation task that assesses long-term context coherence during backtracking. Finally, we adopt **BigCodeBench** (Zhuo et al., 2024) for test-driven program synthesis, evaluating the system's efficiency in handling structured, syntax-heavy repetitions during iterative debugging.

Models and RL algorithms. We employ **Qwen2.5-7B-Instruct** (Yang et al., 2024) and **Qwen3-8B** (Yang et al., 2025) as representative backbones. For policy optimization, we utilize two distinct algorithms: **PPO** (Schulman et al., 2017) and **GRPO** (Shao et al., 2024). This selection is strategic as they impose contrasting distributional constraints on the rollout phase: PPO typically processes diverse prompts sequentially, whereas GRPO mandates the parallel instantiation of G replicas for a single prompt. This diversity allows us to rigorously benchmark RANA's robustness across fundamentally different execution paradigms.

Metrics. Given that RANA is specifically designed to optimize the rollout phase, our primary evaluation metric is **end-to-end rollout duration**, which serves as the definitive indicator of the system's effective acceleration. Beyond duration metric, we further conduct a granular analysis of the **long-tail latency**. This secondary assessment allows us to isolate RANA's efficacy in compressing the execution time of straggler tasks, which typically constitute the primary bottleneck in multi-turn generation.

4.2 MAIN RESULTS

To rigorously evaluate the efficacy of our system, we conducted comprehensive experiments across all permutations of the aforementioned datasets, models, and algorithms on eight RTX 4090 with 48G VRAM each, benchmarking RANA against the vanilla baseline. To ensure statistical reliability and minimize the impact of transient system fluctuations, we executed each experimental setting for 10 iterations. The results reported herein represent the average values derived from these runs, offering a robust assessment of the system's performance under stable conditions.

The quantitative results from our primary experiments are summarized in Table 1. On average, RANA delivers a $\mathbf{2.09}\times$ improvement in end-to-end rollout throughput. The system exhibits its peak efficiency in Maze environment using Qwen2.5-7B-Instruct, achieving a maximum speedup of $\mathbf{3.74}\times$.

Table 1: End-to-end rollout makespan comparison across different models, algorithms, and datasets. Lower is better; **bold** indicates the best result within each backbone.

		Qwen2.5-7B-instruct		Qwen3-8B	
Algorithm	Dataset	w/o RANA	w/ RANA	w/o RANA	w/ RANA
PPO	GSM8K	233.14	**124.77**	274.90	**135.45**
	Maze	415.97	**111.24**	447.48	**257.27**
	BigCodeBench	374.17	**160.16**	409.83	**224.07**
GRPO	GSM8K	514.54	**290.47**	470.27	**333.91**
	Maze	634.99	**229.91**	1695.02	**785.14**
	BigCodeBench	758.27	**344.76**	632.51	**516.77**

4.3 Comparison with model-based speculative decoding

To contextualize RANA against state-of-the-art *model-based* speculative decoders, we additionally benchmark against **EAGLE-3**. The EAGLE-3 draft model is initialized from publicly released, pretrained checkpoints and held *frozen* during RL training—i.e., only the target policy is updated, mirroring the standard deployment pattern of pretrained speculative-decoding kernels. Because EAGLE-3 introduces an additional draft-model footprint that exceeds the 48 GB VRAM of our main RTX 4090 testbed, this set of experiments is conducted on a separate cluster of **8× NVIDIA A100 (80 GB)** GPUs, while all other experiments in this paper use RTX 4090. Detailed server configurations for both setups are listed in Appendix C.1.

Table 2: Comparison with EAGLE-3 on A100 platform. End-to-end rollout makespan on $8\times$ A100 (80 GB); lower is better. EAGLE-3 uses pretrained, frozen draft checkpoints. **Bold** indicates the best result within each backbone.

		Qwen3-1.7B			Qwen3-8B		
Algorithm	Dataset	w/o RANA	w/ RANA	EAGLE-3	w/o RANA	w/ RANA	EAGLE-3
PPO	GSM8K	151.02	**80.26**	277.21	260.12	**158.60**	171.14
	Maze	533.69	**368.55**	886.79	499.63	**353.27**	385.99
GRPO	GSM8K	532.33	**326.76**	1093.73	1016.34	**768.04**	841.75
	Maze	2097.57	**1196.62**	3213.88	1587.94	**1209.98**	1217.55

As shown in Table 2, RANA consistently outperforms EAGLE-3 across every configuration where both methods complete, despite incurring no extra parameters or training cost for the drafter. We attribute this gap to the inherent mismatch between EAGLE-3's frozen, pretrained draft model and the continuously updated RL policy, which erodes its acceptance rate as training progresses. The result validates RANA's central design choice of a model-free, history-driven drafter that adapts to the evolving policy without any retraining.

4.4 Analysis of long-tail latency

Given that RANA is fundamentally architected to mitigate the straggler effect in asynchronous rollouts, assessing average throughput alone is insufficient to fully characterize its structural advantages. To rigorously validate the system's efficacy in addressing the bottleneck, we conduct further analysis of the tail latency, specifically focusing on the **P90 metric**. This metric quantifies the execution duration required to resolve the final 10% of tasks, which typically constitute the computational long tail.

We conducted this evaluation using the **Qwen3-8B** model optimized via the **PPO** algorithm across all benchmark datasets. For each rollout session, we tracked the task completion timeline and highlight the duration of the tail phase, from the completion of 90th percentile task to the end of rollout.

As illustrated in Figure 3, the results demonstrate a dramatic improvement in tail performance. RANA successfully compresses the execution time of the long-tail phase by up to **74%** compared to the vanilla baseline. This substantial reduction provides strong empirical evidence that RANA effectively aligns speculative resources with the most computationally demanding tasks. By specifically targeting

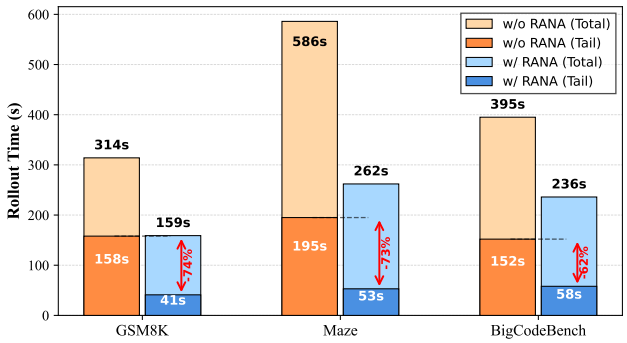


Figure 3: Impact of RANA on long-tail latency (P90). The chart compares the time required to complete the final 10% of tasks using Qwen3-8B with PPO.

and accelerating these stragglers, RANA eliminates the primary blocking factor in the pipeline, thereby driving the significant boost in overall RLHF rollout efficiency reported in Table 1.

4.5 Ablation Studies and Analyses

To validate the individual contributions of RANA's components, we performed an ablation study on the Maze dataset using Qwen3-8B with GRPO (Table 3).

Table 3: Ablation study results on the Maze dataset using Qwen3-8B with GRPO.

Setting	E2E Rollout Time	Speedup
w/o RANA	1695.02	1.00 ×
+ SD	915.91	1.85 ×
+ Controller	889.88	1.90 ×
+ Scheduler	785.14	2.16 ×
Scheduler Only	1583.89	1.07 ×

Component contributions. The introduction of Speculative Decoding (SD) serves as the primary acceleration engine, delivering a substantial $\mathbf{1.85}\times$ speedup. The addition of the Controller further refines this to $\mathbf{1.90}\times$. The full system, integrating the dynamic Scheduler, achieves the peak performance of $\mathbf{2.16}\times$, demonstrating the cumulative benefit of the complete architecture.

Scheduler efficiency. Notably, when only scheduler is enabled in the experiment, we still gets a $\mathbf{1.07}\times$ improvement over the baseline. This result is significant as it isolates the Entropy-Aware Scheduler's impact, proving that its task reordering strategy effectively mitigates the long-tail effect and enhances throughput even in non-speculative scenarios.

5 Conclusion

In this paper, we introduced RANA, a high-performance system designed to tackle the computational inefficiencies inherent in multi-turn RLHF rollouts for reasoning-intensive agents. We identified three critical opportunities for optimization: contextual continuity, hardware regime sensitivity, and entropy decay. RANA systematically exploits these properties through a cohesive architecture comprising a Trie-based N-gram Speculative Decoding mechanism, a Load-Adaptive Controller, and an Entropy-Aware Scheduler.

Our approach shifts the paradigm from blind speculation to a strategic, structure-aware acceleration. RANA effectively eliminates the long-tail bottleneck that plagues conventional pipelines. Empirical results validate the efficacy of this design, demonstrating up to a $\mathbf{3.74}\times$ speedup in end-to-end throughput.

AI USE STATEMENT

In this work, we used generative AI tools for assisting with running experiments, and for polishing the writing. We have not used generative AI tools for research idea generation or for writing code. We have reviewed all AI-assisted work: all experimental results were checked by the authors, and all AI-polished text was reviewed by the authors to ensure that it faithfully reflects our original content. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Alfred V Aho and Margaret J Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

Chi-Chih Chang, Siqi Zhu, Zhichen Zeng, Haibin Lin, Xin Liu, Jiaxuan You, Mohamed S Abdelfattah, Ziheng Jiang, and Xuehai Qian. Srt: Accelerating reinforcement learning via speculative rollout with tree-structured cache. In *NeurIPS 2025 Workshop on Efficient Reasoning*, 2025.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

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

Rongxin Cheng, Kai Zhou, Xingda Wei, Siyuan Liu, Mingcong Han, Mingjing Ai, Yeju Zhou, Baoquan Zhong, Wencong Xiao, Rong Chen, et al. Fast llm post-training via decoupled and fastest-of-n speculation. *arXiv preprint arXiv:2511.16193*, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of llm inference using lookahead decoding. In *Forty-first International Conference on Machine Learning*, 2024.

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

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Jingkai He, Tianjian Li, Erhu Feng, Dong Du, Qian Liu, Tao Liu, Yubin Xia, and Haibo Chen. History rhymes: Accelerating llm reinforcement learning with rhymerl. *arXiv preprint arXiv:2508.18588*, 2025.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 19274–19286, 2023.

Yubo Li, Xiaobin Shen, Xinyu Yao, Xueying Ding, Yidi Miao, Ramayya Krishnan, and Rema Padman. Beyond single-turn: A survey on multi-turn interactions with large language models. *arXiv preprint arXiv:2504.04717*, 2025.

Bingshuai Liu, Ante Wang, Zijun Min, Liang Yao, Haibo Zhang, Yang Liu, Anxiang Zeng, and Jinsong Su. Spec-rl: Accelerating on-policy reinforcement learning via speculative rollouts. *arXiv preprint arXiv:2509.23232*, 2025.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

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

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

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

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

Bao Shu, Yan Cai, Jianjian Sun, Chunrui Han, En Yu, Liang Zhao, Jingcheng Hu, Yinmin Zhang, Haoran Lv, Yuang Peng, et al. Thinking by doing: Building efficient world model reasoning in llms via multi-turn interaction. *arXiv preprint arXiv:2511.23476*, 2025.

Ruiyi Wang and Prithviraj Ammanabrolu. A practitioner's guide to multi-turn agentic reinforcement learning. In *First Workshop on Multi-Turn Interactions in Large Language Models*, 2025.

Siqi Wang, Hailong Yang, Junjie Zhu, Xuezhu Wang, Yufan Xu, and Depei Qian. Rlhfspec: Breaking the efficiency bottleneck in rlhf training via adaptive drafting. *arXiv preprint arXiv:2512.04752*, 2025a.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. In *The Twelfth International Conference on Learning Representations*, 2024.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025b.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhongzhi Li, Xiangyuan Xue, Yijiang Li, et al. The landscape of agentic reinforcement learning for llms: A survey. *arXiv preprint arXiv:2509.02547*, 2025.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37: 62557–62583, 2024.

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

Yinmin Zhong, Zili Zhang, Bingyang Wu, Shengyu Liu, Yukun Chen, Changyi Wan, Hanpeng Hu, Lei Xia, Ranchen Ming, Yibo Zhu, et al. Optimizing {RLHF} training for large language models with stage fusion. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pp. 489–503, 2025b.

Yifei Zhou, Song Jiang, Yuandong Tian, Jason Weston, Sergey Levine, Sainbayar Sukhbaatar, and Xian Li. Sweet-rl: Training multi-turn llm agents on collaborative reasoning tasks. *arXiv preprint arXiv:2503.15478*, 2025.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*, 2024.

A Algorithm Details

Algorithm 1 provides the full pseudocode of the trie-based N-gram speculative drafting procedure described in Section 3.1.

Algorithm 1 N-gram speculative drafting via suffix trie

Input: Context suffix s, Reference Trie $\mathcal{T}$, Budget γ
Output: Speculative draft sequence $\tilde{x}$
$u \leftarrow \mathcal{T}$.root
for token t in s **do**
  while $t \notin u$.children **and** $u \neq \mathcal{T}$.root **do**
    $u \leftarrow \pi(u)$
  end while
  if $t \in u$.children **then**
    $u \leftarrow u$.children$[t]$
  end if
end for
$\tilde{x} \leftarrow \emptyset$
for $j = 1$ **to** γ **do**
  $v \leftarrow \arg\max_{k \in u.\text{children}} \text{Count}(u.\text{children}[k])$
  $\tilde{x} \leftarrow \tilde{x} \oplus v$
  $u_{\text{next}} \leftarrow u$.children$[v]$
  if Depth$(u_{\text{next}}) > L$ **then**
    $u \leftarrow \pi(u_{\text{next}})$
  else
    $u \leftarrow u_{\text{next}}$
  end if
end for
return $\tilde{x}$

B Theoretical Analysis

B.1 Amortized $O(1)$ retrieval via state persistence

This subsection expands the complexity argument summarized in Section 3.1. We address the cost on two temporal scales: intra-step sliding window and inter-step state persistence.

First, within a single speculative step, generating a draft of length γ naively requires re-scanning the updated window from scratch, leading to $O(L \times \gamma)$ complexity. To alleviate the complexity, we leverage failure pointers to implement a sliding window. If the transition from the current node u via token $\tilde{x}_j$ leads to a full match (depth $L + 1$), the system follows the failure link $\pi(u)$ once to shift the window; otherwise, it continues deepening the trie. This reduces the intra-step complexity to $O(L + \gamma)$.

We further optimize the *inter-step* dynamics to decouple retrieval latency from both the context window L and the draft budget γ. Standard approaches reset the trie traversal to the root after every verification step, incurring a repetitive $O(L)$ scanning cost. In contrast, RANA implements *State Persistence*. Let u_t denote the trie node pointer at the beginning of decoding step t. After the drafter proposes a sequence $\tilde{x}$ and the target model verifies and accepts the prefix $\tilde{x}_{1:\alpha}$ (where $\alpha \leq \gamma$), the starting state for the next step, u_{t+1}, is mathematically guaranteed to be the node reached by transitioning from u_t via $\tilde{x}_{1:\alpha}$. Since this path has already been traversed during the drafting phase of step t, the update operation $u_{t+1} \leftarrow \text{Transition}(u_t, \tilde{x}_{1:\alpha})$ incurs no additional search cost.

Consequently, the total number of state transitions, including forward edges and failure links, performed by the drafter over a complete generation session of length T is bounded by $2T$. The forward transitions exactly equal T (one per generated token), and by the aggregate property of the Aho-Corasick automaton, the total number of failure link traversals cannot exceed the total forward transitions. Thus, the complexity of amortized retrieval is given as:

$$\mathcal{C}_{\text{amortized}} = \frac{\sum \text{Ops}}{T} \leq \frac{2T}{T} = O(1). \tag{7}$$

This proves that RANA's retrieval overhead is asymptotically constant per token, strictly independent of the growing context history L or the speculative depth γ.

B.2 Roofline analysis of the load-adaptive controller

To design a theoretically grounded controller, we analyze the hardware boundaries of speculative decoding. Let $T_{op}(N)$ denote the latency of a forward pass with N tokens. Following the Roofline model, latency is constrained by either memory bandwidth β_{mem} or peak compute π_{peak}:

$$T_{op}(N) = \max\left(\frac{W_{mem}}{\beta_{mem}}, \frac{N \cdot P}{\pi_{peak}}\right). \tag{8}$$

Defining the speedup ratio S as the gain in effective throughput given an acceptance rate α and speculation step γ, we derive:

$$S = \frac{T_{op}(B) \cdot (1 + \alpha\gamma)}{T_{op}(B(1+\gamma))}. \tag{9}$$

This formulation reveals two distinct operational regimes governed by the runtime batch size B: In the memory-bound regime with low workload, when bandwidth dominates, latency is invariant to small increases in token count ($T_{op}(B) \approx T_{op}(B(1+\gamma))$). The verification overhead is masked, yielding a strict speedup $S \approx 1 + \alpha\gamma > 1$. Conversely, in the compute-bound regime with high workload, when compute dominates, latency scales linearly with workload. The speedup degenerates to $S \approx \frac{1+\alpha\gamma}{1+\gamma}$. Since $\alpha < 1$ in realistic settings, this results in *negative acceleration* ($S < 1$), where verification costs outweigh speculative gains.

This dichotomy identifies a critical *saturation point* B_{sat}. Consequently, our controller aims to dynamically approximate the optimal step size: $\gamma_t^* = \gamma_{\text{default}}$ if $B_t < B_{sat}$ else 0.

B.3 Optimality of the entropy-aware scheduler

We formalize the scheduling objective as maximizing the expected speculative speedup $\mathbb{E}[S_{\text{tail}}]$ during the memory-bound phase. Given the inverse correlation $\alpha \propto -H$, maximizing speedup is structurally equivalent to minimizing the aggregate entropy of the tail workload $\mathcal{Q}_{\text{tail}}$:

$$\max_{\mathcal{Q}_{\text{tail}}} \mathbb{E}[S_{\text{tail}}] \iff \min_{\mathcal{Q}_{\text{tail}}} \sum_{S_i \in \mathcal{Q}_{\text{tail}}} H(S_i). \tag{10}$$

By greedily flushing high-entropy tasks during the saturated phase (as formulated in Section 3.3), our strategy mathematically ensures that the residual set $\mathcal{Q}_{\text{tail}}$ achieves this entropy lower bound. Consequently, RANA guarantees a strictly higher effective acceptance rate than FCFS scheduling, i.e., $\mathbb{E}[\alpha_{\text{RANA}}] > \mathbb{E}[\alpha_{\text{FCFS}}]$, ensuring speculation is applied precisely where the theoretical gain is maximized.

C EXPERIMENTAL SETUP

This appendix provides the full experimental configuration that supports the results in the main text. We include (i) the SGLang serving parameters used to instantiate every variant of the rollout engine (§C.1), (ii) sampling-time hyper-parameters and the entropy-aware priority signal (§C.2), (iii) the multi-turn interaction protocol that drives both PPO and GRPO style rollouts (§C.3), and (iv) the verbatim system / user / feedback prompts of every evaluated environment (§C.5). Hardware: $8\times$ NVIDIA A100 (80 GB) GPUs, one model replica per GPU (TP$=1$) / $8\times$ NVIDIA RTX 4090 (48 GB) GPUs, one model replica per 2 GPUs (TP$=2$)

C.1 SGLANG SERVING CONFIGURATIONS

We benchmark three rollout-engine configurations that share the same model weights and chat template but differ in their scheduling policy and speculative-decoding stack.

(C1) Vanilla FCFS. A baseline server with neither speculative decoding nor priority scheduling; requests are served in first-come-first-served order.

(C2) RANA. A draft-model-free speculative decoder based on n-gram lookup, combined with the SGLang priority scheduler. Default priority for fresh requests is set to a large constant (10^9); the priority of subsequent turns is overridden by an entropy-aware signal (§C.2, Eq. 12).

(C3) EAGLE-3. Speculative decoding with a dedicated tree-based draft model (ANGELSLIM/QWEN3-{1.7B,8B}_EAGLE3). Following the recommended recipe we use `num_steps`$=6$, `topk`$=10$ and `num_draft_tokens`$=32$. Because the EAGLE-3 KV-pool reclamation path in the FlashInfer attention backend exhibits a known memory-leak bug, we explicitly switch to the FA3 backend for this configuration.

A consolidated parameter list is given in Table 4. All values not shown follow the SGLang upstream defaults (commit-pinned to the version used in our experiments).

Table 4: SGLang server arguments for the three rollout-engine configurations. *All* runs share dtype `bfloat16`, attention backend `fa3` and serve a single model replica per GPU (A100) or serve a model replica per 2 GPUs with TP size=2 (4090).

Argument	**C1: FCFS**	**C2: RANA**	**C3: EAGLE-3**
`schedule-policy`	`fcfs`	`fcfs` †	`fcfs`
`enable-priority-scheduling`	— (off)	on	— (off)
`priority-preempt-threshold`	—	10^9	—
`speculative-algorithm`	— (off)	`NGRAM`	`EAGLE3`
`speculative-draft-model`	—	—	`AngelSlim/Qwen3-{1.7B,8B}_eagle3`
`speculative-num-steps`	—	—	6
`speculative-eagle-topk`	—	—	10
`speculative-num-draft-tokens`	—	4	32
`cuda-graph-max-bs`	default	default	2
`attention-backend`	`fa3`	`fa3`	`fa3`
`max-running-requests` (GSM8K)	128	128	256
`max-running-requests` (Maze)	64	64	64
`mem-fraction-static` (1.7B)	0.55	0.55	0.70
`mem-fraction-static` (8B)	0.65	0.65	0.85

† Under priority scheduling (`enable-priority-scheduling`), `fcfs` only specifies the tie-break rule among requests with identical priority.

C.2 SAMPLING HYPER-PARAMETERS AND ENTROPY-AWARE PRIORITY

For every chat-completion call we use the OpenAI-compatible endpoint `/v1/chat/completions` with the parameters shown in Table 5. We always request top-10 logprobs in order to estimate token-level uncertainty.

Table 5: Sampling parameters (identical across all environments and engine configurations).

Parameter	Value
`temperature`	0.7
`max_tokens`	2048
`logprobs`	`true`
`top_logprobs`	10
`model`	`Qwen/Qwen3-1.7B` / `Qwen/Qwen3-8B`
`sampling_params.session_id`	task id (PPO) / group id (GRPO)

Truncated entropy. Given the top-K logprobs $\{\ell_1, \ldots, \ell_K\}$ (K=10) returned by the server, we form the re-normalised distribution $\tilde{p}_i = e^{\ell_i} / \sum_j e^{\ell_j}$ and compute the *truncated* Shannon entropy and varentropy

$$H_t \;=\; -\sum_{i=1}^{K} \tilde{p}_i \log \tilde{p}_i, \qquad \sigma^2_{H_t} \;=\; \sum_{i=1}^{K} \tilde{p}_i \big(\log \tilde{p}_i + H_t\big)^2. \tag{11}$$

The per-turn entropy is the arithmetic mean over generated tokens, $\bar{H} = \frac{1}{T}\sum_t H_t$.

Priority signal. For C2 we map $\bar{H}$ of the previous turn to a scheduling priority used at the next turn:

$$\text{priority} \;=\; \big\lfloor 10^4 \cdot \bar{H} \big\rfloor. \tag{12}$$

The first turn uses the constant priority 10^9. In SGLang, larger priority values are served first; thus high-uncertainty turns are scheduled preferentially over low-uncertainty turns, mirroring our partial-online hypothesis (high-entropy steps benefit most from on-policy data).

C.3 Multi-turn Rollout Protocol

Each task is rolled out as a multi-turn dialogue capped at $R_{\max}$ turns. At every turn the engine receives the full conversation history, returns one assistant message, and an environment-specific verifier produces either (i) a *terminal* success signal or (ii) a textual feedback message that becomes the next user turn. A trajectory ends when the verifier reports a pass with score ≥ 0.95 or when $R_{\max}$ is reached.

PPO mode. Each task is rolled out exactly once (*group_size*$= 1$). The `session_id` sent to the engine is the task index, so KV / radix entries are not shared across tasks.

GRPO mode. We replicate each task $K = 4$ times. All replicas of the same task share a common `session_id` (the original task index), which lets SGLang reuse the radix-cache prefix across replicas while still producing K stochastic rollouts.

Qwen3 thinking mode. For GSM8K we append the suffix " `/no_think`" to every user turn (the standard switch documented in the Qwen3 chat template) so that the model emits a direct answer without internal chain-of-thought tokens. For the Maze environment we leave the default "thinking" behaviour enabled, because spatial reasoning benefits from explicit deliberation.

C.4 Datasets

Statistics of the two evaluation suites are summarised in Table 6.

Table 6: Evaluation datasets.

Dataset	#Tasks	Avg. chars	Source / construction
GSM8K-Hard-Mixed	512	≈ 256	curated subset of GSM8K test split, hard problems
Maze-Easier	63	7×7 grid	generated with Prim's algorithm (§C.5)

Maze generation. Every maze is a 7×7 grid built with Prim's algorithm, starting from a random odd cell. The agent "P" is placed at a navigable corner and the goal "X" at the maximally-distant reachable cell (BFS frontier). Walkable cells are marked "o", walls "$\star$". The maximum-step budget per episode is 100. The agent may emit at most 5 atomic moves per turn (see §C.5.2).

C.5 Environment Prompts

We list verbatim the system prompt, the initial user turn and the verifier feedback messages for every environment. Curly placeholders `{...}` are runtime substitutions.

C.5.1 GSM8K (rule-verified arithmetic)

System prompt.

```
You are a math expert.  You are given a question and you
need to solve it step by step.  You should rethinking
carefully if user point out your answer is wrong.  Put your
final answer in the format of #### <answer>, while <answer>
MUST contain digits and decimal point ONLY.
```

Initial user turn. The raw GSM8K question text, optionally with the suffix " `/no_think`".

Feedback message.

- *No format match*: "`Your response doesn't contain answer in corrent format.  You need to reflect on your answer and MUST provide your answer in format "#### <Answer>".`"
- *Correct*: "`Your response is correct!`"
- *Incorrect*: "`Your response is incorrect!  You need to reflect on your answer and try again.`"

C.5.2 Maze (multi-step pathfinding)

System prompt.

```
You are an expert Pathfinding Agent using A* logic.  Your
task is to navigate a 2D Maze Grid from a Start Point to a
Goal Point.
### 1.  COORDINATE SYSTEM (CRITICAL)
- The maze is a 2D grid.
- Coordinates are formatted as:  (ROW, COLUMN).
- ROW: Vertical index (0 is Top, increases downwards).
- COLUMN: Horizontal index (0 is Left, increases
rightwards).
- Moves:
    'move up' -> (Row - 1, Col)
    'move down' -> (Row + 1, Col)
    'move left' -> (Row, Col - 1)
    'move right' -> (Row, Col + 1)
### 2.  MAP LEGEND
- 'P': Current Player Position
- 'X': Goal
- 'o':  Empty space (Walkable)
- '*':  Wall (Impassable)
### 3.  THINKING PROCESS (MANDATORY)
Before outputting actions, you MUST perform a 'Spatial
Check':
1.  Identify the current coordinate (Row, Col).
2.  Check the content of all 4 adjacent cells.
3.  Discard any move that hits a Wall ('*').
4.  Choose the best move that brings you closer to the Goal
```

```
'X' without hitting a wall.
5.  If you hit a wall in the previous turn, DO NOT repeat
the same move.
### 4.  OUTPUT FORMAT
Output the movement sequence enclosed in <action> tags,
separated by semicolons.  Limit to 5 steps at a time to
ensure accuracy.
Example:  <action>move right;move down</action>
```

Initial user turn (template).

```
Start Navigation Task.
Maze Layout (Rows 0-{H-1}, Cols 0-{W-1}):
{maze_with_row_col_indices}
Current State:
- Player Position:  (Row:  {py}, Col:  {px})
- Goal Position:  (Row:  {gy}, Col:  {gx})
- Obstacle Positions (impassable):  {(r, c), ...}
Legend:  'P' player, 'X' goal, 'o' empty, '*' wall.
Analyse the grid specifically around ({py},{px}) before
moving.  Find a path to 'X'.
```

Feedback message (template, abridged). After every turn the verifier returns one of three structured messages:

1. **NO VALID ACTIONS**: when no `<action>` block is parsed, the feedback restates the position, the four adjacent cell types and a 5×5 local map view around the player.
2. **COLLISION**: when the agent attempts to enter a wall, the feedback identifies the offending move, restates the position, lists the four adjacent cells and the local map, and instructs the agent not to repeat the move.
3. **SUCCESS (in progress)**: lists the previous executed actions, new position, step count, four adjacent cells and the local map view.

A trajectory terminates with "`Congratulations!  You have reached the goal in {n} steps!`" on success or "`Max steps ({100}) reached without finding the goal.`" on exhaustion.

C.5.3 BIGCODEBENCH

System prompt. Identical structure to HumanEval, augmented with explicit guidance about library imports and edge cases.

Initial user turn. "`Task: Complete the following programming task.\n\n{prompt}\n\nPlease provide your complete implementation (including any necessary imports) wrapped in <code></code> tags.`"

Verifier. The last `<code>...</code>` block is extracted and executed in a sandboxed Python interpreter (60 s wall-clock, 2 GB memory) against the dataset's gold tests. Pass iff all asserts succeed.