MART: MULTI-AGENT HYPERPARAMETER TUNING FOR LLM REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM reinforcement learning (RL) is computationally intensive, and well-chosen hyperparameters can improve both training efficiency and model performance. However, parameter tuning is highly dependent on expert knowledge and repeated trial-and-error experiments, incurring considerable human and computational overhead. Recent agent-based approaches have demonstrated the potential of automating system configuration optimization for LLM training and serving. However, extending such approaches to RL requires accounting for its distinct execution and learning dynamics: rollout and policy-update phases impose different memory and compute demands, while response lengths evolve during training, changing the workload and complicating configuration evaluation. Moreover, effective tuning must consider both execution efficiency and the effects of learning hyperparameters on training stability and task performance. To address these challenges, we propose **MART**, a multi-agent framework with human-in-the-loop review for hyperparameter tuning in RL post-training. Guided by phase-wise resource measurements and training feedback, specialized agents propose and assess candidate configurations. A human reviewer approves each selected candidate before execution or rejects it with feedback for subsequent revision. The search proceeds in two stages: short-horizon trials identify system configurations that improve end-to-end throughput while retaining a prescribed GPU memory margin; longer-horizon trials then optimize learning hyperparameters with the system configuration fixed. During learning tuning, online training-health assessment supports early termination of trials exhibiting persistent anomalies. Experiments on GRPO-based mathematical reasoning show that, relative to baseline configurations, MART improves throughput averaged over updates 3–5 by 13.5% and 7.7% for Qwen3-8B, and by 69.5% and 91.3% for Qwen3-0.6B-Base, under single-/dual-node deployments, respectively. Across AMC23, OlympiadBench, AIME24, and AIME25, MART improves mean pass@1 by 1.98–2.51 percentage points for Qwen3-8B and 0.74–0.81 percentage points for Qwen3-0.6B-Base.

1 INTRODUCTION

Reinforcement learning (RL) has become a key component of large language model (LLM) post-training for alignment and reasoning (Ouyang et al., 2022; Bai et al., 2022; Guo et al., 2025). However, RL post-training requires the repeated execution of response generation, reward computation, and policy updates, which results in high computational overhead. Its efficiency and effectiveness are also highly sensitive to configuration choices (Eimer et al., 2023). System parameters, such as tensor parallelism (TP) (Shoeybi et al., 2019), pipeline parallelism (PP) (Huang et al., 2019), and micro-batch size, affect memory usage and training throughput, while learning parameters, such as learning rate and KL regularization coefficient, influence training stability and final task performance. Selecting these configurations in practice still relies heavily on expert knowledge and repeated trial-and-error, resulting in substantial human and computational overhead.

RL post-training poses additional challenges for automated tuning, as Figure 1 illustrates. First, its execution phases exhibit heterogeneous resource behavior. In the representative Qwen3-8B training step shown in Figure 1(a), GPU memory usage differs substantially across rollout generation, actor and reference log-probability computation, and policy update. The resource bottleneck is therefore phase-dependent, making aggregate resource measurements insufficient for diagnosing and com-

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.

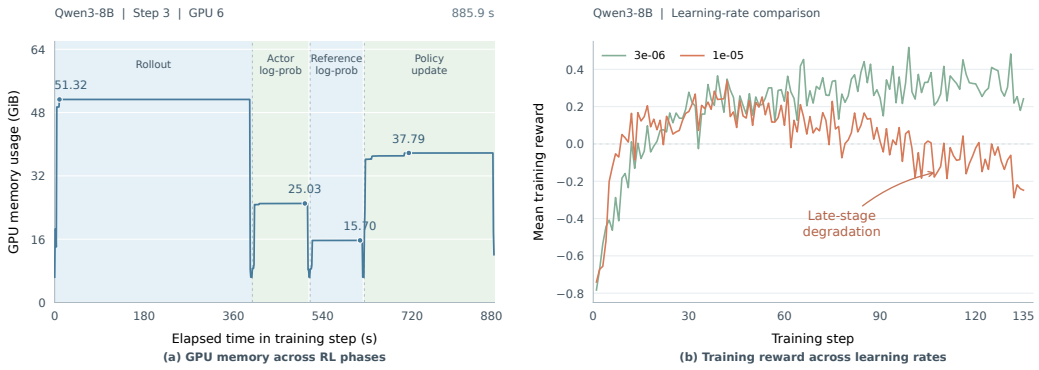

Figure 1: Motivating challenges in RL post-training tuning, illustrated with Qwen3-8B. (a) GPU memory usage on a single GPU during the third training step varies substantially across execution phases. (b) The run with learning rate 10^{-5} improves early but later exhibits sustained reward degradation, whereas the 3×10^{-6} run maintains higher rewards later in training, illustrating the limitations of short-horizon evaluation. Each reward curve represents one run. All curves show raw measurements without smoothing.

paring system configurations. Second, learning configurations cannot always be reliably evaluated from short training windows. Figure 1(b) shows that a learning rate of 10^{-5} improves reward early in training but later exhibits sustained degradation, whereas 3×10^{-6} maintains a more stable trajectory. Moreover, the generation workload itself evolves as the policy changes the response-length distribution, causing memory pressure and execution cost to vary over time. These characteristics imply different evidence requirements and evaluation horizons for system and learning hyperparameters: short trials can provide actionable resource and throughput measurements, whereas learning configurations require longer trajectories to assess training behavior and task performance.

Existing approaches address different parts of this problem. Classical and multi-fidelity HPO methods improve candidate selection and evaluation-budget allocation (Bergstra & Bengio, 2012; Snoek et al., 2012; Jamieson & Talwalkar, 2016; Li et al., 2018; Falkner et al., 2018), and recent work further reduces the cost of HPO for LLM RL (Chen et al., 2026a). Distributed training and RL systems optimize parallelism and cross-phase execution (Zheng et al., 2022; Li et al., 2022; Liu et al., 2024a; Mei et al., 2024; Sheng et al., 2025), while LLM-based approaches increasingly use optimization trajectories, profiling information, and historical experiments to guide configuration decisions (Chen et al., 2022; Yang et al., 2024; Liu et al., 2024b; Chen et al., 2026b; Ding et al., 2026). However, these directions largely focus on either learning-hyperparameter search or system optimization. A practical challenge is to coordinate system-configuration search and learning-hyperparameter search when they require different runtime evidence and evaluation horizons in RL post-training.

To address these challenges, we propose **MART**, a multi-agent framework with human-in-the-loop review for hyperparameter tuning in LLM RL post-training. MART decomposes the tuning into two sequential stages according to the evidence required by different parameter groups. The *system tuning* stage uses short trials and phase-wise runtime measurements to optimize end-to-end throughput while preserving GPU-memory headroom. The selected system configuration is then fixed, and the *learning tuning* stage uses longer trials to optimize learning hyperparameters based on training trajectories and task performance.

The two stages share an evidence-driven decision–execution pipeline. Specialized agents propose and assess candidate configurations, diagnose failed trials, and summarize historical experience, while deterministic modules enforce configuration and resource constraints. Before execution, a human reviewer approves the selected configuration or rejects it with feedback that guides subsequent revisions by the Proposal Agent. During learning tuning, a Train Health Agent further evaluates rule-triggered anomalies using training trajectories and historical evidence, supporting early termination of persistently anomalous trials. In this way, MART continuously converts runtime observations, human feedback, and previous trial outcomes into evidence for subsequent configuration decisions.

The main contributions of this work are:

1. We propose **MART**, a multi-agent framework with human-in-the-loop review that separates RL post-training hyperparameter tuning into short-horizon system tuning and longer-horizon learning tuning, according to their different evidence and evaluation requirements.
2. We develop an evidence-driven tuning loop that combines phase-wise resource measurements, structured trial history, agent reasoning, deterministic constraint checking, and human approval or rejection feedback. During learning tuning, online training-health assessment supports early termination of persistently anomalous trials.
3. We evaluate **MART** on GRPO-based mathematical reasoning tasks with Qwen3-8B and Qwen3-0.6B-base under both single-node (8-GPU) and dual-node (16-GPU) deployments. Across these four settings, MART improves throughput averaged over updates 3–5 by 7.7–91.3% and mean test accuracy across four benchmarks by 0.74–2.51 percentage points relative to the corresponding baselines.

2 RELATED WORK

2.1 HYPERPARAMETER OPTIMIZATION FOR RL

Hyperparameter optimization (HPO) aims to identify effective configurations under a limited evaluation budget. Classical approaches include random search (Bergstra & Bengio, 2012) and Bayesian optimization (Snoek et al., 2012), while multi-fidelity methods reduce search cost by allocating different resources to candidate configurations. Successive Halving (Jamieson & Talwalkar, 2016), Hyperband (Li et al., 2018), and BOHB (Falkner et al., 2018) progressively concentrate computation on promising candidates. Hyperparameter selection is particularly consequential in RL, where configuration choices can substantially affect optimization behavior and performance (Eimer et al., 2023). More recently, JF-HPO (Chen et al., 2026a) targets LLM RL using model size and training budget as fidelity dimensions, together with early stopping and checkpoint reuse to reduce evaluation cost. MART addresses a complementary challenge: system and learning hyperparameters require different evaluation signals and trial lengths in RL post-training. We therefore separate their tuning into short-horizon system tuning and longer-horizon learning tuning.

2.2 DISTRIBUTED TRAINING AND RL SYSTEM OPTIMIZATION

Automated distributed-training systems search parallelism configurations to improve training efficiency. Alpa (Zheng et al., 2022) automates inter- and intra-operator parallelism, AMP (Li et al., 2022) searches model-parallel strategies with hardware awareness, Galvatron (Miao et al., 2022) and Aceso (Liu et al., 2024a) optimize hybrid parallelism through search or iterative bottleneck alleviation. For RL workloads, ReaLHF (Mei et al., 2024) uses parameter reallocation and profiling-based cost estimation to adapt execution strategies, while HybridFlow (Sheng et al., 2025) improves cross-phase execution through model resharding and device mapping. These systems primarily optimize execution plans and resource allocation. MART instead treats system configuration as part of a broader hyperparameter-tuning workflow for RL post-training.

2.3 LLM- AND AGENT-DRIVEN AUTOMATIC TUNING

Language models are increasingly used as optimizers that use previous evaluations to guide subsequent decisions. OptFormer (Chen et al., 2022) predicts promising hyperparameter configurations from optimization trajectories, while OPRO (Yang et al., 2024) prompts an LLM with previous solutions and scores to iteratively propose improved candidates. AgentHPO (Liu et al., 2024b) uses specialized agents to refine hyperparameters from historical experiments. For system optimization, OptiCo (Chen et al., 2026b) applies collaborative agents to bottleneck analysis, strategy generation and validation, and failure diagnosis, while PROMPTS (Ding et al., 2026) combines profiling information with optimization knowledge to recommend configurations for LLM training and serving. MART extends agent-based tuning to RL post-training by grounding decisions in runtime evidence, trial history, deterministic constraints, and online training-health feedback.

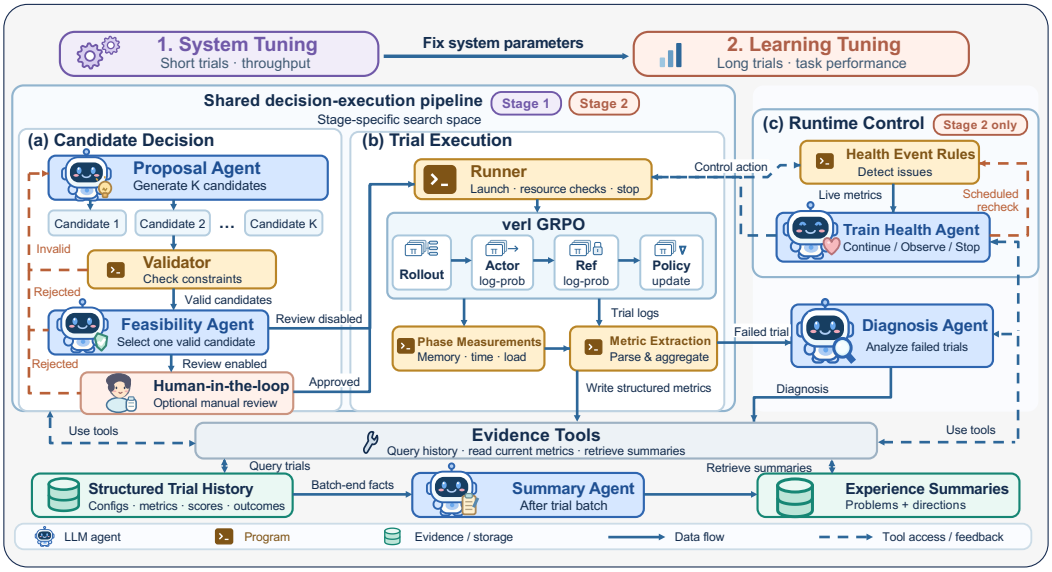

Figure 2: Overview of the MART framework.

3 METHOD

3.1 FRAMEWORK

Figure 2 presents MART, a multi-agent framework that separates RL hyperparameter tuning into two sequential stages according to the different evaluation requirements of system and learning parameters. In the *system tuning* stage, MART evaluates system configurations using short trials to select a configuration that improves end-to-end training throughput while satisfying a prescribed GPU memory margin. After system tuning, the selected system configuration is fixed. MART then enters the *learning tuning* stage, where longer trials are used to evaluate learning hyperparameters based on training trajectories and task performance. This separation avoids spending long training budgets on system configurations.

Despite using different evaluation horizons and objectives, the two stages share the same decision–execution pipeline, consisting of *Candidate Decision* and *Trial Execution*. Candidate Decision generates candidate configurations, validates their constraints, and selects a configuration for execution, while Trial Execution launches the corresponding RL training run and converts its runtime behavior into structured trial evidence. MART further enables *Runtime Control* only during learning tuning, which allows early termination of persistently anomalous trials.

MART closes the tuning loop by transforming each executed trial into reusable evidence for subsequent decisions. Structured Trial History stores trial-level facts, including configurations, resource measurements, training metrics, evaluation scores, and outcomes. After each batch, the Summary Agent consolidates recurring problems and effective or ineffective adjustment directions into Experience Summaries. The Proposal, Feasibility, Diagnosis, and Train Health Agents access trial-level records or summarized experience through agent-specific tools in the shared *Evidence Tools* interface (see Appendix B.8 for access permissions). Deterministic modules, including configuration validation, metric extraction, and others, are used when explicit constraints require enforcement. Together, these components form an evidence-driven feedback loop in which observations from previous trials guide subsequent configuration proposals, assessments, and runtime decisions.

3.2 CANDIDATE DECISION

The candidate decision process consists of candidate generation, constraint validation, and feasibility assessment and selection, performed by the Proposal Agent, Validator, and Feasibility Agent, respectively. It uses the search space and constraints of the current stage together with historical

trial records. Tools are available to query parameter semantics and historical metrics and to estimate memory requirements. When the preceding trial fails or is terminated early due to training health issues, the corresponding failure diagnosis or training-health assessment is also made available to inform the next decision.

MART adopts a human-in-the-loop (HITL) workflow by default, allowing users to review the candidate selected by the Feasibility Agent before execution. Users can either approve the candidate for Trial Execution or reject it with feedback to guide candidate revision. The framework also supports an automated mode, denoted as **MART/no-HITL**, in which the selected candidate is forwarded directly to Trial Execution without human review.

Proposal Agent: The Proposal Agent generates a set of K candidates using stage-specific context and historical trial evidence, with tools available to retrieve additional information when needed. During system tuning, phase-wise memory and timing measurements help identify potential bottlenecks and guide parameter adjustments. During learning tuning, training trajectories and task evaluation results guide learning hyperparameter adjustments aimed at improving task performance and training stability. Each candidate i is specified by a reference trial r_i, parameter changes Δ_i, a rationale for the changes, and their expected effects on relevant metrics. Its complete configuration is constructed programmatically as

$$x_i = \mathrm{Apply}(x_{r_i}, \Delta_i). \tag{1}$$

Different candidates can start from different historical trials and propose reasonable improvement directions addressing distinct potential bottlenecks. Details are prorided in Appendix B.2.

Validator: The Validator checks the reconstructed candidate configurations for consistency between the declared reference values and the recorded configuration, stage-specific edit permissions, parameter ranges, structural compatibility, and duplicate configurations. For example, parallelism configurations must be compatible with the available GPU count, and system parameters must remain fixed during learning tuning. Once at least $K_{\min}$ candidates have passed validation, the validated candidates are forwarded to the Feasibility Agent.

Feasibility Agent: The Feasibility Agent reassesses validated candidates using historical trial evidence to compare their expected benefits and risks. During system tuning, it uses phase-wise memory and timing measurements, together with memory estimates, to assess memory feasibility under the prescribed headroom requirement and potential gains in end-to-end training throughput. During learning tuning, it uses historical training trajectories and task evaluations to assess potential performance gains and training-health risks. The agent either selects one validated candidate without modifying its configuration or rejects all candidates. Details in Appendix B.3.

Rejection Feedback: If fewer than $K_{\min}$ candidates pass validation or the Feasibility Agent rejects all candidates, the rejection reasons are returned to the Proposal Agent to guide candidate revision. In MART, feedback accompanying human rejection is likewise returned to the Proposal Agent. The feedback guides subsequent rounds of candidate generation. In MART, retries are unlimited, and the process continues until a candidate receives human approval. In MART/no-HITL, this process repeats until a candidate is selected or the retry budget $R_{\max}$ is exhausted.

3.3 Trial Execution and Feedback

Trial execution converts selected configurations into structured evidence for subsequent tuning decisions. Phase-wise resource measurements and end-to-end throughput support system tuning, while training trajectories and task evaluations support learning tuning. Failure diagnoses and batch-level summaries provide additional feedback for subsequent candidate proposals and assessment.

Runner: The Runner launches each trial with the training-step budget assigned to the current tuning stage. During execution, phase-wise monitoring records GPU memory usage, GPU utilization, and execution time across rollout, actor and reference log-probability computation, and policy updates. Metric extraction aggregates these measurements and parses training logs to obtain end-to-end throughput, training metrics, and available task evaluation results.

During system tuning, MART requires GPU memory headroom of at least R_{safe} to leave capacity for possible increases in memory demand during subsequent learning trials. A system-tuning trial is terminated if its observed runtime headroom falls below this threshold. The configuration, extracted

metrics, memory-check results, and trial outcome are stored in Structured Trial History. Training-health control during learning tuning is described in Section 3.4.

Diagnosis agent: When a trial fails, the Diagnosis Agent analyzes error logs and phase-wise measurements to identify the likely failure type and affected execution phase. It provides supporting evidence and a confidence assessment to guide subsequent candidate revision. For early stopping caused by training health issues, the system reuses the Train Health Agent's assessment and supporting evidence. Details are provided in Appendix B.4.

Summary agent: After each trial batch, the Summary Agent compares the executed trials with their respective reference experiments, summarizes the discovered issues and the effective and ineffective adjustment directions by tuning stage, and consolidates the evidence from similar attempts. The summary preserves the associated trial IDs and is written into the experience memory for on-demand retrieval by subsequent candidate decision-making. Detailed in Appendix B.5.

3.4 Runtime Control

Runtime control is employed exclusively during the learning tuning stage. We use a rule-triggered agent to identify potential risks and support early termination of persistently anomalous trials, thereby reducing the computational resource consumption caused by anomalous training.

Health-Event Rules: We use a deterministic monitor to track reward, KL, and policy entropy during learning tuning. Let r_t, k_t, and h_t denote the corresponding observations at update t. When the required observations are available, the monitor evaluates the following rules:

- ***Reward degradation.*** To reduce the influence of short-term fluctuations, we compute the moving-average reward over w updates and its historical maximum:

$$\bar{r}_t = \frac{1}{w} \sum_{i=t-w+1}^{t} r_i, \qquad b_t = \max_{w \le s \le t} \bar{r}_s. \tag{2}$$

  Over the latest L updates ($L > w$), the rule requires an absolute drawdown of at least δ_r and no increase between consecutive window means greater than η_r:

$$(b_t - \bar{r}_t \ge \delta_r) \ \wedge \ \left(\max_{j \in \{t-L+w+1,\ldots,t\}} (\bar{r}_j - \bar{r}_{j-1}) \le \eta_r \right). \tag{3}$$

  This rule captures both sustained declines and low plateaus following a decline.

- ***Abrupt KL change.*** The rule requires both the absolute and relative changes to meet their respective thresholds:

$$(|k_t - k_{t-1}| \ge \delta_k) \ \wedge \ \left(\frac{|k_t - k_{t-1}|}{\max\left(|k_{t-1}|, 10^{-12}\right)} \ge \rho_k \right). \tag{4}$$

- ***Abrupt entropy decline.*** The rule is satisfied when the relative entropy decrease reaches ρ_h:

$$\frac{h_{t-1} - h_t}{\max\left(|h_{t-1}|, 10^{-3}\right)} \ge \rho_h. \tag{5}$$

The monitor enables event generation after T_w warm-up updates and enforces a cooldown interval of C updates between events to limit repeated triggering.

Train health agent: The Train Health Agent combines health events, current training metric trajectories, and relevant historical trials to assess whether anomalies persist and whether there is evidence of recovery. It recommends `continue`, `observe`, or `stop`, together with a supporting rationale. When `observe` is selected, the system schedules a follow-up assessment after the specified number of updates, incorporating newly available observations. The Runner executes approved stop actions. Detailed in Appendix B.6.

Table 1: Test accuracy (%) on the mathematical reasoning benchmarks. Throughput (tokens/s/GPU) is averaged over steps 3–5, total training time (s) and total training tokens (M) are also reported for single-node (8 GPUs) and dual-node (16 GPUs) deployments.

Method	AMC23 ↑	OlympiadBench ↑	AIME24 ↑	AIME25 ↑	Avg. ↑	Throughput ↑	Time ↓	Total Tokens
Qwen3-8B — 1 × 8 GPUs								
baseline	68.91	44.88	24.58	21.67	40.01	731	136041	854.5
MART/no-HITL	67.97	44.79	**26.67**	24.79	41.06	790	**135845**	895.4
MART	**71.88**	**46.52**	26.25	**25.42**	**42.52**	**830**	149271	982.5
Qwen3-8B — 2 × 8 GPUs								
baseline	67.34	44.47	24.17	22.92	39.73	724	69123	856.7
MART/no-HITL	64.84	44.92	**27.29**	**23.96**	40.25	**804**	**66869**	899.5
MART	**68.91**	**48.34**	26.04	23.54	**41.71**	780	71208	897.5
Qwen3-0.6B-base — 1 × 8 GPUs								
baseline	28.12	16.74	1.46	1.46	11.95	1793	27202	397.5
MART/no-HITL	**28.44**	**18.65**	2.29	1.25	12.66	2983	16787	399.1
MART	27.97	17.77	**2.92**	**2.08**	**12.69**	**3040**	**15041**	389.2
Qwen3-0.6B-base — 2 × 8 GPUs								
baseline	25.47	**17.70**	2.29	0.83	11.57	1585	14090	387.7
MART/no-HITL	**29.22**	17.47	**2.92**	1.04	**12.66**	2395	**10100**	400.5
MART	27.97	17.39	2.50	**1.67**	12.38	**3032**	11460	636.4

4 EXPERIMENT

4.1 SETTINGS

Experiment setting: We evaluate our framework on mathematical reasoning tasks using Qwen3-8B and Qwen3-0.6B-base (Yang et al., 2025) under both single-node and dual-node settings. All experiments are conducted on MetaX C550 GPUs and use experience summaries with the same cutoff date. We adopt Group Relative Policy Optimization (GRPO) (Shao et al., 2024) for post-training, using verl 0.7, with vLLM 0.13.0 (Kwon et al., 2023) serving as the rollout backend and Megatron 0.15.2 (Shoeybi et al., 2019) as the training backend. We compare our method with repository configurations verified to run on the C550 platform.

Agent Configuration: All five agents use DeepSeek V4.1 Flash (Xu et al., 2026) as the backbone model. Detailed agent configurations, including parameter settings and their correspondence to the notation, are provided in Appendix B.1.

Evaluation Protocol: During tuning, all trials are conducted on MATH, while MATH500 (Hendrycks et al., 2021; Lightman et al., 2023) is used as the validation set for hyperparameter selection. Using the best configuration, we additionally evaluate the checkpoint on AMC 2023, OlympiadBench (He et al., 2024), AIME 2024, and AIME 2025, reporting pass@1 estimated from 16 samples per problem. Evaluation uses a temperature of 1.0, top-p of 1.0, no top-k filtering, and a maximum response length of 2,048 tokens. Training throughput is averaged over steps 3–5 for all configurations, skipping potential warm-up in the first two steps and limiting the influence of token workload changes induced by learning hyperparameters.

4.2 MAIN RESULTS

Table 1 reports test accuracy, throughput, training time and tokens, while Figure 3 presents validation accuracy against cumulative training time, from the corresponding training logs. Within each model and deployment setting, all configurations use the same training-step budget and sample budget.

Training Efficiency: Both MART and MART/no-HITL improve early-step throughput across model scales and deployment settings, consistent with the system-tuning objective. However, changes to learning hyperparameters can alter training trajectories and cumulative token workloads, so throughput gains need not translate into proportional reductions in total training time.

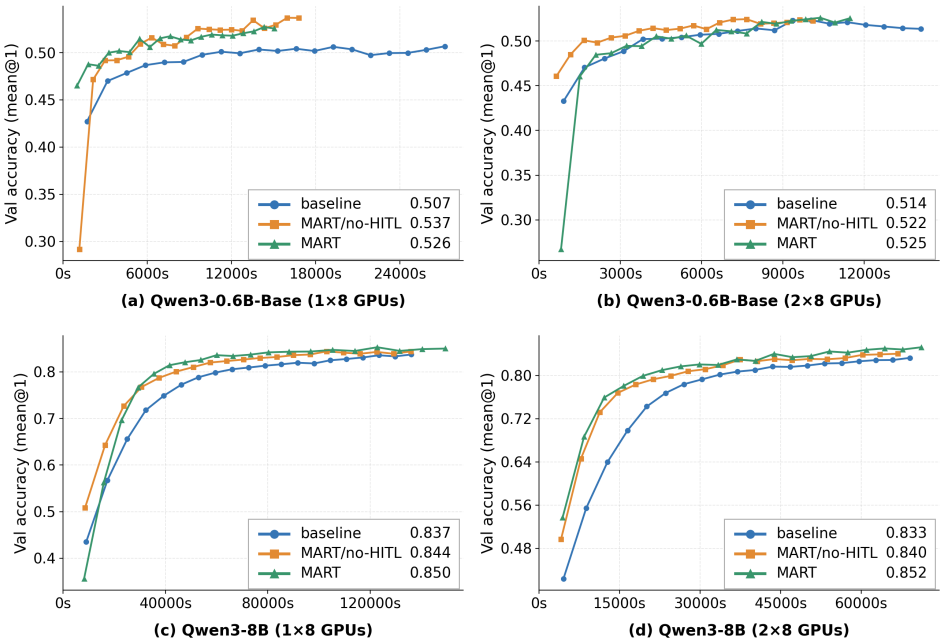

Figure 3: Validation accuracy over time with the same total number of training steps.

Nevertheless, both variants can reduce training time even with larger cumulative token workloads. For Qwen3-0.6B-base, MART reduces total training time by 44.7% and 18.7% under single-node and dual-node deployments, respectively, with the dual-node reduction achieved despite processing 64.1% more tokens than the baseline. For Qwen3-8B under dual-node deployment, MART/no-HITL reduces training time by 3.3% while processing 5.0% more tokens. These results highlight the practical value of the selected configurations and support MART's two-stage strategy of improving execution efficiency through system tuning before evaluating learning hyperparameters over longer training trajectories.

Mathematical Reasoning Performance: Both MART and MART/no-HITL improve average test accuracy over the baseline across four settings. For Qwen3-8B, MART/no-HITL achieves gains of 1.05 and 0.52 percentage points under single/dual-node deployments, respectively, while MART achieves gains of 2.51 and 1.98 percentage points. For Qwen3-0.6B-base, single/dual-node gains are 0.74/0.81 percentage points for MART and 0.71/1.09 for MART/no-HITL.

For Qwen3-8B, MART achieves higher mean test accuracy than MART/no-HITL, while both variants obtain similar average scores for Qwen3-0.6B-base, with MART/no-HITL performing better under the dual-node deployment. These improvements indicate that MART's benefits extend beyond execution throughput to aggregate reasoning performance. Together with the training-efficiency results, they show MART can identify configurations that improve average test accuracy while reducing total training time.

Validation Performance During Training: As shown in Figure 3, both MART and MART/no-HITL generally achieve higher validation accuracy than the baseline at comparable elapsed times during the early and intermediate stage of training. This indicates that their benefits translate into better performance within a given time budget. For Qwen3-8B, these accuracy gaps narrow later during training, while both variants retain higher final validation accuracy than the baseline across all four settings. The curves thus illustrate how the selected configurations balance learning performance and execution efficiency over the course of training.

4.3 Ablation Experiments

Figure 4(a) illustrates the trial sequence during the learning-tuning stage. Figure 4(b) and Table 2 examine contributions of human-in-the-loop, phase-wise memory information and experience summaries, and the effect of switching the agent backbone to a different model. All experiments use Qwen3-0.6B-base on a single node with eight GPUs.

Table 2: Ablation study.

Method	AMC23 ↑	OlympiadBench ↑	AIME24 ↑	AIME25 ↑	Avg. ↑	Throughput ↑	Time (s) ↓	Total Tokens (M)
MART	27.97	17.77	2.92	2.08	**12.69**	**3040**	**15041**	389.2
MART/no-HITL	**28.44**	**18.65**	2.29	1.25	12.66	2983	16787	399.1
MART/noHITL Qwen3.8	27.81	18.34	2.29	**2.29**	12.68	2914	15396	377.5
MART/noHITL/noPhase	26.56	17.52	1.67	1.04	11.70	2650	18537	421.4
MART/noHITL/noExp	25.62	17.06	**3.96**	1.04	11.92	2734	17086	402.2
baseline	28.12	16.74	1.46	1.46	11.95	1793	27202	397.5

Learning Tuning: Figure 4(a) illustrates a learning-stage run in which not every configuration adjustment improves performance. This example illustrates how MART can identify improved configurations through iterative exploration and evaluation, even when individual trials do not consistently yield gains.

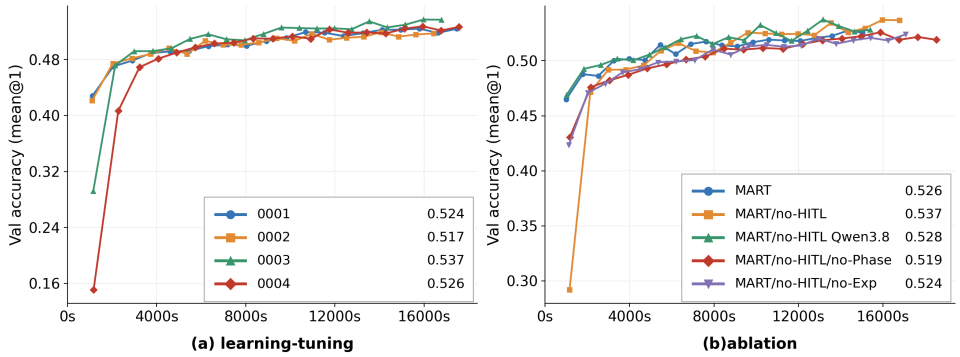

Figure 4: Validation accuracy versus training time under the same total training-step budget: (a) learning-tuning trials; (b) ablation settings.

Human review and agent backbone: Removing human review slightly lowers mean test accuracy and early-step throughput in this setting. The Qwen3-8B results in Table 1 show a larger accuracy gap of 1.46 percentage points in favor of MART under both deployments. Within MART/no-HITL, replacing the default backbone with Qwen3.8 (Qiu et al., 2026) yields similar mean test accuracy (12.68% vs. 12.66%) and early-step throughput, suggesting limited sensitivity to backbone choice in this setting.

Experience summaries: Removing experience summaries from MART/no-HITL reduces mean test accuracy by 0.74 percentage points and early-step throughput by 8.3%, alongside lower final validation accuracy. These results suggest that summarized tuning experience provides useful guidance for subsequent configuration decisions.

Phase-wise memory information: Replacing phase-wise GPU memory measurements with aggregate information reduces early-step throughput from 2,983 to 2,650 tokens/s/GPU, an 11.2% decrease relative to MART/no-HITL. This result supports the value of phase-wise memory information in guiding system configuration selection for higher end-to-end training throughput.

5 Conclusion

We presented MART, a multi-agent framework that separates RL hyperparameter tuning into system and learning stages, using phase-wise runtime measurements and trial feedback to guide configuration search. Experiments on mathematical reasoning demonstrate improvements in training throughput and average test accuracy over baseline configurations. MART achieves higher average test accuracy in most settings, suggesting that human feedback remains valuable and that fully automated tuning leaves room for improvement. Moreover, the current search space covers a limited set of common system and learning parameters, including TP, MBS, and learning rate. Future work will focus on improving autonomous tuning decisions and expanding the search space to include additional optimization strategies, such as activation recomputation.

AI USE STATEMENT

We used generative AI tools to assist with creating the visual depiction of the MART architecture, editing the manuscript's language and grammar, generating code for plotting experimental results, and identifying potentially relevant literature. Generative AI tools were not used to formulate the core ideas of MART, design its conceptual architecture, or design the key experiments. The LLM agents within MART are part of the method described in this paper and are distinct from the authoring assistance disclosed here. The authors reviewed the AI-assisted material and take responsibility for the final content, results, and claims.

REPRODUCIBILITY STATEMENT

The Method section describes MART's two-stage search procedure, candidate validation, memory checks, and training-health control. The Experiment section specifies the models, datasets, hardware, software versions, and evaluation protocol. The appendix documents trial budgets, editable parameter groups, health-event thresholds, agent-tool permissions, and representative prompt and decision examples. These sections document the workflow and experimental protocol used in this study.

ETHICS STATEMENT

This work evaluates hyperparameter tuning for GRPO-based post-training on mathematical reasoning tasks. The human-in-the-loop variant reviews proposed training configurations rather than evaluating the human reviewers themselves. The experiments do not intentionally use personal or sensitive data. MART requires accelerator compute and external LLM API calls; we report training time, throughput, and API-token usage to make these resource demands visible. The broader effects of more efficient post-training depend on the downstream model and application, and our empirical claims are limited to the models, benchmarks, and hardware evaluated here.

REFERENCES

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

Minping Chen, Bowen Xiao, Du Liang, Chuxuan Zeng, and Zeyi Wen. Efficient hyperparameter optimization for LLM reinforcement learning. In *ACL*, pp. 27540–27552, San Diego, California, United States, 2026a. doi: 10.18653/v1/2026.acl-long.1271.

Sheng Chen, Zhe Tang, Weixing Zhang, Fei Yang, Yuanyuan Wang, Tianlin Li, and Yang Liu. OptiCo: Adaptive distributed training optimization via collaborative agent reasoning. In *ACL*, pp. 27847–27867, 2026b.

Yutian Chen, Xingyou Song, Chansoo Lee, Zi Wang, Richard Zhang, David Dohan, Kazuya Kawakami, Greg Kochanski, Arnaud Doucet, Marc'Aurelio Ranzato, et al. Towards learning universal hyperparameter optimizers with transformers. *Advances in Neural Information Processing Systems*, 35:32053–32068, 2022.

Yuran Ding, Ruobing Han, Xiaofan Zhang, and Xinwei Chen. Prompts: Performance optimization via multi-agent planning for llm training and serving. *Proceedings of Machine Learning and Systems*, 8:1010–1039, 2026.

Theresa Eimer, Marius Lindauer, and Roberta Raileanu. Hyperparameters in reinforcement learning and how to tune them. In *International conference on machine learning*, pp. 9104–9149. PMLR, 2023.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *ICML*, pp. 1437–1446. PMLR, 2018.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv*, 2025.

Luo Renjie He, Chaoqun et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *ACL*, pp. 3828–3850, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. GPipe: Efficient training of giant neural networks using pipeline parallelism. In *NeurIPS*, 2019.

Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial intelligence and statistics*, pp. 240–248. PMLR, 2016.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Dacheng Li, Hongyi Wang, Eric Xing, and Hao Zhang. Amp: Automatically finding model parallel strategies with heterogeneity awareness. *Advances in Neural Information Processing Systems*, 35: 6630–6639, 2022.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of machine learning research*, 18(185):1–52, 2018.

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. *arXiv preprint arXiv:2305.20050*, 2023.

Guodong Liu, Youshan Miao, Zhiqi Lin, Xiaoxiang Shi, Saeed Maleki, Fan Yang, Yungang Bao, and Sa Wang. Aceso: Efficient parallel dnn training through iterative bottleneck alleviation. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pp. 163–181, 2024a.

Siyi Liu, Chen Gao, and Yong Li. Large language model agent for hyper-parameter optimization. *arXiv preprint arXiv:2402.01881*, 2024b.

Zhiyu Mei, Wei Fu, Kaiwei Li, Guangju Wang, Huanchen Zhang, and Yi Wu. Realhf: Optimized rlhf training for large language models through parameter reallocation. *arXiv preprint arXiv:2406.14088*, 2024.

Xupeng Miao, Yujie Wang, Youhe Jiang, Chunan Shi, Xiaonan Nie, Hailin Zhang, and Bin Cui. Galvatron: Efficient transformer training over multiple gpus using automatic parallelism. *arXiv preprint arXiv:2211.13878*, 2022.

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.

Zihan Qiu, Zekun Wang, Xiao Li, Yanpeng Li, Yang Xu, Yixuan Wang, Huaqing Zhang, Rui Men, Bochao Mao, Chengruidong Zhang, et al. On the design of qwen3. 8-next architecture: Evaluation, efficiency, and training stability. *arXiv preprint arXiv:2608.30320*, 2026.

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.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. In *arXiv preprint arXiv:1909.08053*, 2019.

Jasper Snoek, Hugo Larochelle, and Ryan Adams. Practical bayesian optimization of machine learning algorithms. *NeurIPS*, 25, 2012.

Anyi Xu, B Li, Bangcai Lin, Bing Xue, BingCheng Xian, Bingzheng Xu, Bochao Wu, Bowei Zhang, Boyi Deng, CC Yu, et al. Deepseek-v4. 1-flash: Pushing the limits of kv cache compression. *arXiv preprint arXiv:2609.19969*, 2026.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *arXiv*, 2025.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, volume 2024, pp. 12028–12068, 2024.

Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P Xing, et al. Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 559–578, 2022.

A Supplementary Experiments and Token Usage Analysis

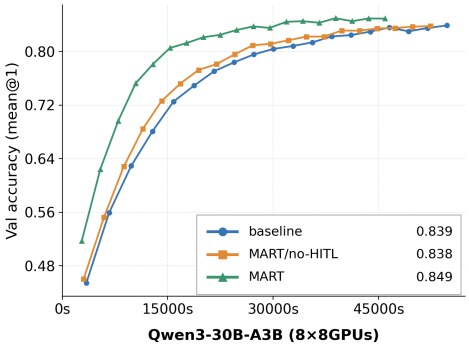

Figure 5: Validation accuracy over time for the MoE model at the same training step.

Table 3: Training efficiency of the Qwen3-30B-A3B MoE model using 8×8 GPUs. Throughput (tokens/s/GPU) is averaged over steps 3–5; total training time (s) and total training tokens (M) are reported.

Method	Throughput ↑	Time (s) ↓	Total Tokens (M)
baseline	239	56870	839.9
MART/no-HITL	269	52340	847.9
MART	289	47796	849.2

A.1 Additional MoE Results

We further evaluate MART on the Qwen3-30B-A3B mixture-of-experts (MoE) model using 8×8 GPUs. Figure 5 shows the validation accuracy versus training time. MART achieves a validation accuracy of 84.9%, outperforming the baseline's 83.9% while requiring less time. As shown in Table 3, MART= reduces the runtime from 56,870 s to 47,796 s, a reduction of approximately 16.0%, and increases throughput from 239 to 289 tokens/s/GPU. These results demonstrate that MART also improves training efficiency and validation performance in the MoE setting.

A.2 Token Usage Statistics and Analysis

Table 4: LLM API token usage by agent for one complete experiment, comprising 10 tuning trials: six for system tuning and 4 for learin tuning.

Tokens	Proposal	Feasibility	Diagnosis	Summary	Train Health	Total
Input	911,970	399,183	–	19,006	117,646	1,447,805
Output	179,046	137,589	–	15,032	14,483	346,150
Total	1,091,016	536,772	–	34,038	132,129	1,793,955

LLM API token usage. Table 4 reports the LLM API token usage of each agent over one complete experiment, comprising six system-tuning and four learning-tuning trials. The experiment consumes approximately 1.79M tokens overall, including 1.45M input tokens and 0.35M output tokens. The Proposal and Feasibility agents account for most of the usage, consuming approximately 1.09M and 0.54M tokens, respectively, while the Summary and Train Health agents consume approximately 34K and 132K tokens.

B Implementation Details

B.1 Agent Configuration

This appendix documents the configuration of MART's tuning workflow, including trial budgets, candidate decision-making, and online training-health monitoring.

Trial budgets. The planned tuning budget consists of six system-tuning trials, each targeting five training updates, and four learning-tuning trials, each targeting 100 updates.

TRIAL_BUDGETS

system_trial_updates
Update budget for each system-tuning trial.
Value: `5`

learning_trial_updates
Update budget for each learning-tuning trial.
Value: `100`

system_trial_count
Planned number of system-tuning trials.
Value: `6`

learning_trial_count
Planned number of learning-tuning trials.
Value: `4`

Candidate decision. In each round, the Proposal Agent generates three to five candidates, with at least three valid candidates required for feasibility review. Each candidate may change at most 13 parameters. With manual review disabled, the selected candidate proceeds automatically to execution. Proposal and validation may take up to six rounds, and the agents receive up to three historical trial records as context. A system trial must retain at least 4096 MiB of free GPU memory to be accepted after measurement.

CANDIDATE_DECISION

manual_review_enable
Human in the loop is initiated if and only if `manual_review_enable` is true.
Value: `false`

min_proposal_candidates
Minimum proposal count and minimum valid candidates for feasibility review.
Value: `3` **Symbol:** $K_{\min}$

max_proposal_candidates
The maximum number of proposals is denoted by K, with

$$K \in \texttt{range(max_parameter_changes, max_parameter_changes+1)}.$$

Value: `5` **Symbol:** K

max_parameter_changes
Maximum number of parameter changes in one candidate.
Value: `13`

max_validation_rounds
Maximum proposal/validation rounds, including the initial round.
Value: `6` **Symbol:** $R_{\max}$

history_prompt_trials
Limit on trial-history entries injected into Agent context.
Value: `3`

throughput_memory_reserve_mib
Recommended candidate memory thresholds from the Proposal Agent, in MiB.
Value: `8192`

resource_memory_reserve_mib
Stricter free-memory floor for accepting a system trial after measurement, in MiB.
Value: `4096` **Symbol:** R_{safe}

Health-event rules. During learning tuning, deterministic rules monitor reward, KL, and policy entropy for sustained reward degradation, abrupt KL changes, and entropy declines. The detailed configuration is as follows:

HEALTH_EVENT_RULES

health_kl_metric
Training metric used as the KL observation.
Value: `"actor/kl_loss"` **Symbol:** k_t

health_reward_metric
Training metric used as the reward observation.
Value: `"critic/rewards/mean"` **Symbol:** r_t

health_reward_trend_steps
Valid reward observations spanned by the trend check.
Value: `5` **Symbol:** L

health_reward_window_size
Reward moving-average window length.
Value: `3` **Symbol:** w

health_reward_trend_min_drawdown
Minimum absolute drawdown from the best reward window mean.
Value: `0.15` **Symbol:** δ_r

health_reward_trend_tolerance
Maximum adjacent window-mean increase allowed as non-improvement.
Value: `0.01` **Symbol:** η_r

health_kl_change_ratio_threshold
Relative KL-change threshold; combined with the absolute threshold.
Value: `0.50` **Symbol:** ρ_k

health_kl_change_absolute_threshold
Minimum absolute KL change; covers increases and decreases.
Value: `0.02` **Symbol:** δ_k

health_entropy_drop_ratio_threshold
Minimum relative decrease in policy entropy.
Value: `0.30` **Symbol:** ρ_h

health_warmup_updates
Generate rule events only when the update index exceeds this value.
Value: `2` **Symbol:** T_w

health_agent_cooldown_updates
Shared rule-event cooldown in updates; scheduled reviews are independent.
Value: `5` **Symbol:** C

B.2 Details of Proposal Agent

The Proposal Agent uses a shared prompt template instantiated with stage-specific context. The orchestrator supplies the current stage, editable parameters, fixed parameter values, reference trials, observed metrics, and available tools. Appendix B.7 documents the parameter partition and provides concrete context examples. Here, *fixed* refers to the current stage: system tuning freezes learning parameters, while learning tuning freezes hardware parameters. Parameters outside both editable sets remain fixed throughout the workflow. For brevity, the prompt below is condensed and edited for presentation and is not a verbatim reproduction of the runtime prompt.

Prompt of Proposal Agent

You are the parameter-selection and experimental-design specialist in an automated verl 0.7 GRPO tuning system. Generate a small, diverse set of evidence-based candidates. Each candidate must test a clear causal hypothesis and may inherit its configuration from a different recorded trial.

Current Stage: {`CURRENT_STAGE`}

Immutable Model, Hardware, and Workload Context:
{IMMUTABLE_CONTEXT}
Parameters Fixed in This Stage:
{FIXED_PARAMETERS}
Default Reference Trial:
{DEFAULT_REFERENCE}
Compact Reference History:
{COMPACT_REFERENCE_HISTORY}
Editable Parameters:
{EDITABLE_PARAMETERS}
Current Values of Editable Parameters:
{EDITABLE_PARAMETER_VALUES}
Hard Constraints:
{CONSTRAINTS}
Most Recent Failure Diagnosis:
{DIAGNOSIS}
Available Tools:
{AVAILABLE_TOOLS}

Objectives:

1. During system tuning, improve end-to-end training throughput while preserving memory feasibility.
2. During learning tuning, improve MATH evaluation accuracy while keeping hardware parameters frozen. Use reward, KL, entropy, and related signals as supporting evidence for training health and causal diagnosis.

Reference and Evidence Rules: *[Detailed reference and evidence rules omitted]*
Candidate Generation rules and Output:
[Detailed candidate generation rules and the JSON output format are omitted]

Example proposal. The following abridged output comes from a system-tuning repair round after trial 4 exceeded the training-phase memory limit and after the first-round candidate that **disabled use_dynamic_bsz was rejected**. Explanatory text is shortened for presentation.

Example Output of the Proposal Agent: System-Tuning Repair (Abridged)


```
{
  "decision": "modify",
  "reason": "Trial 4 reached a training-memory peak of 63840 MiB,
    exceeding the 57344 MiB acceptance limit. A recent attempt to
    disable use_dynamic_bsz and reduce micro_bsz was rejected. Human
    feedback advised against changing use_dynamic_bsz and suggested
    modifying the topology instead. A PP-only restoration would
    duplicate trial 2, so three distinct repairs are proposed.",
  "candidates": [
    {
      "candidate_id": "repair_pp2_tok8192",
      "reference_trial_id": 4,
      "reference_reason": "Trial 4 is the failed PP=1 configuration
    being repaired.",
      "reason": "Restore PP=2, which has measured support from trial
    2, and reduce the dynamic micro-batch token cap to distinguish
    this repair from a direct revert.",
      "changes": {
      "actor_rollout_ref.actor.megatron.pipeline_model_parallel_size":
     {
```

```
          "from": 1,
          "to": 2,
          "reason": "Restore pipeline splitting to reduce per-GPU
    training memory."
        },
        "actor_rollout_ref.actor.ppo_max_token_len_per_gpu": {
          "from": 16384,
          "to": 8192,
          "reason": "Reduce activation memory while remaining above
    the 3072-token sequence limit."
        }
      },
      "expected_effect": {
        "metric_name": "training_peak_memory_mib",
        "direction": "decrease"
      }
    },
    {
      "candidate_id": "repair_pp1_tok4096",
      "reference_trial_id": 4,
      "reference_reason": "Trial 4 provides the failed PP=1
    configuration.",
      "reason": "Retain PP=1 and test whether a smaller dynamic micro-
    batch token cap resolves the training-memory breach.",
      "changes": {
        "actor_rollout_ref.actor.ppo_max_token_len_per_gpu": {
          "from": 16384,
          "to": 4096,
          "reason": "Reduce activation memory while remaining above
    the 3072-token sequence limit."
        }
      },
      "expected_effect": {
        "metric_name": "training_peak_memory_mib",
        "direction": "decrease"
      }
    },
    {
      "candidate_id": "repair_pp1_tp8",
      "reference_trial_id": 4,
      "reference_reason": "Trial 4 provides the failed PP=1, TP=4
    configuration.",
      "reason": "Retain PP=1 and test whether greater tensor
    parallelism reduces per-GPU model-state memory.",
      "changes": {
      "actor_rollout_ref.actor.megatron.tensor_model_parallel_size": {
          "from": 4,
          "to": 8,
          "reason": "Shard the actor model across eight GPUs."
        }
      },
      "expected_effect": {
        "metric_name": "training_peak_memory_mib",
        "direction": "decrease"
      }
    }
  ]
}
```

B.3 Details of Feasibility Agent

The Feasibility Agent uses a shared prompt template instantiated with stage-specific context. The orchestrator supplies the current stage, validated candidates, reference-trial history, diagnostic feedback, memory safety limits, and available tools. For brevity, the prompt below is condensed and edited for presentation and is not a verbatim reproduction of the runtime prompt.

Prompt of Feasibility Agent

You are the independent candidate reviewer in an automated verl 0.7 GRPO tuning system. Assess the validated candidates using historical evidence, parameter semantics, and resource estimates. Select exactly one candidate by its `candidate_id`, or reject all candidates if none is sufficiently justified. Do not generate, modify, or combine candidates.

Current Stage:
{CURRENT_STAGE}

Validated Candidates:
{CANDIDATES}

Compact Reference History:
{COMPACT_REFERENCE_HISTORY}

Most Recent Failure Diagnosis:
{DIAGNOSIS}

Memory Safety Limits:
{MEMORY_LIMITS}

Available Tools:
{AVAILABLE_TOOLS}

Objectives:

1. During system tuning, use phase-wise resource evidence and memory estimates to assess memory risks and expected end-to-end throughput benefits. Consider cross-phase effects when comparing candidates.
2. During learning tuning, assess candidates using historical MATH validation results, with reward, KL, entropy, and related signals providing supporting evidence of training health. System parameters must remain fixed.

Review and Output:
[Detailed review rules, tool-call requirements, and the JSON output format are omitted.]

Example review. This abridged output reviews three system-tuning repair candidates proposed after trial 4 exceeded the training-phase memory limit. The example retains the candidate verdicts and phase-wise memory estimates; explanatory text is shortened for presentation.

Example Output of the Feasibility Agent: System-Tuning Repair (Abridged)

```
{
  "verdict": "valid",
  "selected_candidate_id": "repair_pp2_tok8192",
  "reason": "The selected repair has the strongest measured support:
    trial 2 with PP=2 recorded a 24194 MiB training peak, whereas
    failed trial 4 with PP=1 reached 63840 MiB. Restoring PP=2
    addresses the observed cause; lowering the token cap to 8192 adds
    headroom.",
  "candidate_reviews": [
    {
      "candidate_id": "repair_pp2_tok8192",
      "verdict": "valid",
      "reason": "Restores the PP=2 topology supported by trial-2
    measurements and lowers the dynamic micro-batch token cap. All
    four phases are estimated within the memory limit.",
      "risks": [
```

```
        "The estimated training peak is uncertain because the
    estimator previously under-predicted the PP=1 peak.",
        "The two-parameter edit makes attribution harder if the repair
     fails."
      ],
      "predicted_memory_mib": {
        "rollout": 52322.0,
        "actor_log_prob": 14494.92,
        "ref_log_prob": 7643.96,
        "training": 44021.84
      }
    },
    {
      "candidate_id": "repair_pp1_tok4096",
      "verdict": "valid",
      "reason": "Reducing the token cap is a coherent single-parameter
     repair, but it retains the PP=1 topology associated with the
    measured breach.",
      "risks": [
        "The predicted reduction may not address a fixed training-
    memory spike."
      ],
      "predicted_memory_mib": {
        "rollout": 52322.0,
        "actor_log_prob": 18321.0,
        "ref_log_prob": 9557.0,
        "training": 47659.41
      }
    },
    {
      "candidate_id": "repair_pp1_tp8",
      "verdict": "invalid",
      "reason": "The TP=8 estimate lacks measured support, some
    configuration constraints remain unverified, and the change may
    reduce end-to-end throughput.",
      "risks": [
        "Training-memory safety and the throughput effect remain
    uncertain."
      ],
      "predicted_memory_mib": {
        "rollout": 52322.0,
        "actor_log_prob": 10693.91,
        "ref_log_prob": 6216.96,
        "training": 45336.82
      }
    }
  ],
  "risks": [
    "The selected candidate's predicted memory is not a measured
    result.",
    "Final safety must be checked by the short-run Resource Gate."
  ],
  "predicted_memory_mib": {
    "rollout": 52322.0,
    "actor_log_prob": 14494.92,
    "ref_log_prob": 7643.96,
    "training": 44021.84
  }
}
```

B.4 Details of Diagnosis Agent

The Diagnosis Agent analyzes failed trials using structured metrics, phase-wise memory measurements, and log evidence. The orchestrator supplies the failed-trial record and available tools. For brevity, the prompt below is condensed and edited for presentation and is not a verbatim reproduction of the runtime prompt. Bracketed omission notes are editorial annotations and are not part of the prompt provided to the agent.

Prompt of Diagnosis Agent

You are the failure-attribution specialist in an automated verl 0.7 GRPO tuning system. Identify the most likely failure type and affected execution phase, and provide supporting evidence and a confidence assessment. Use available tools to obtain additional evidence when needed.
Base your diagnosis on recorded evidence. The phase with the highest measured memory peak does not necessarily identify the failure phase. Report uncertainty when the evidence is insufficient. Do not propose parameter values, redesign experiments, or execute training.

Failed Trial:
{TRIAL}

Available Tools:
{AVAILABLE_TOOLS}

Diagnostic Rules and Output:
[Detailed diagnostic rules, the full failure-label set, and the JSON output format are omitted.]

Example diagnosis. The following abridged output illustrates how the Diagnosis Agent uses a structured failure record and a CUDA traceback to locate an out-of-memory failure in reference log-probability computation.

Example Output of the Diagnosis Agent: Reference Log-Probability OOM (Abridged)

```
{
  "failure_type": "OOM_REF_LOGPROB",
  "training_substage": "ref_log_prob",
  "evidence": [
    "Both the structured failure phase and the recorded memory
    bottleneck identify ref_log_prob.",
    "The CUDA traceback enters compute_ref_log_prob and Megatron's
    float16_to_fp32 conversion; val.float() attempted to allocate
    45.67 GiB with 32.41 GiB free.",
    "Memory samples for ref_log_prob peaked at 79.7% before the
    allocation failure.",
    "Available metrics show no NCCL error, NaN, or training-stability
    collapse."
  ],
  "reason": "The allocation failure occurred during the reference log-
    probability forward pass. The trial changed only the learning rate
    , and the available evidence does not implicate rollout or
    numerical instability; the failure is classified as an OOM in
    ref_log_prob.",
  "confidence": 0.9
}
```

B.5 Details of Summary Agent

The Summary Agent receives structured trial facts, reference-trial records, and run context. It consolidates measured problems and supported adjustment directions by tuning stage, retaining supporting trial IDs for subsequent retrieval. For brevity, the prompt below is condensed and edited for

presentation and is not a verbatim reproduction of the runtime prompt. Bracketed omission notes are editorial annotations and are not part of the prompt provided to the agent.

Prompt of Summary Agent

You summarize the recorded outcomes of a completed or partially completed tuning run. Use only the supplied evidence. Do not propose parameters, call tools, or infer facts that are absent from the input.

Run Facts:
{TRIALS}

Task:
Summarize executed agent-generated trials by tuning stage, comparing each trial with its own reference trial. Identify measured problems and adjustment directions supported as useful or ineffective by the recorded results. Consolidate similar attempts and retain the supporting trial IDs.

Evaluation Principles:

1. During system tuning, assess outcomes primarily through end-to-end throughput and resource feasibility. An improvement in an individual execution phase alone does not establish an overall benefit.
2. During learning tuning, assess outcomes using task evaluation results. Training metrics provide supporting evidence and do not replace missing evaluation results.
3. Both useful and ineffective directions require supporting evidence. Omit inconclusive findings and keep conclusions specific to the supplied run.

Evidence Rules and Output:
[Detailed evidence rules, consolidation requirements, and the JSON output format are omitted.]

Example summary. After a trial batch, the Summary Agent consolidates system- and learning-tuning experience into recurring problems and adjustment directions. The following abridged output retains the trial IDs supporting each observation, so later agents can retrieve the underlying records.

Example Output of the Summary Agent: Trial Experience (Abridged)


```
{
  "run_context": {
    "run_id": "xxxx_xxxx_2026",
    "algorithm": "grpo",
    "model": "Qwen3-8B",
    "platform": "C550",
    "workload": {
      "train_batch_size": 1024,
      "max_prompt_length": 1024,
      "max_response_length": 2048,
      "n_gpus_per_node": 8,
      "nnodes": 2
    }
  },
  "hardware": {
    "problems": [
      {
        "problem": "PP=2 limited data-parallel width and introduced
    pipeline bubbles.",
        "trial_ids": [2]
      },
      {
        "problem": "Rollout became the binding memory and scheduler
    constraint.",
```

```
        "trial_ids": [3, 4]
      },
      {
        "problem": "An undersized rollout log-prob micro-batch limited
     throughput.",
        "trial_ids": [5, 6]
      }
    ],
    "useful_directions": [
      {
        "direction": "Removing pipeline parallelism while reducing per
    -GPU token ceilings improved throughput and actor MFU.",
        "trial_ids": [2]
      },
      {
        "direction": "Increasing rollout memory allocation and
    scheduler concurrency improved throughput within resource limits
    .",
        "trial_ids": [3, 4]
      },
      {
        "direction": "Moderately enlarging the rollout log-prob micro-
    batch helped; further doubling added little benefit.",
        "trial_ids": [5, 6]
      }
    ],
    "ineffective_directions": []
  },
  "stability": {
    "problems": [
      {
        "problem": "Entropy declined while KL loss rose across
    training windows.",
        "trial_ids": [8]
      },
      {
        "problem": "Small KL and clipping values suggested unused
    optimization capacity while validation accuracy was still
    improving.",
        "trial_ids": [10]
      }
    ],
    "useful_directions": [
      {
        "direction": "A moderate learning-rate increase improved
    terminal validation accuracy and reward relative to its reference
    .",
        "trial_ids": [10]
      }
    ],
    "ineffective_directions": [
      {
        "direction": "Changing the KL-loss estimator to mse reduced
    terminal validation accuracy relative to its reference.",
        "trial_ids": [9]
      }
    ]
  }
}
```

B.6 Details of Train Health Agent

The Train Health Agent reviews health events and scheduled follow-ups during learning tuning. The orchestrator supplies the current event, recent trial history, and available tools. The agent uses current training trajectories and historical evidence to assess anomaly persistence and signs of recovery. For brevity, the prompt below is condensed and edited for presentation and is not a verbatim reproduction of the runtime prompt. Bracketed omission notes are editorial annotations and are not part of the prompt provided to the agent.

Prompt of Train Health Agent

You are the independent training-health reviewer in an automated verl 0.7 GRPO tuning system. Assess the currently running learning-tuning trial and recommend `continue`, `observe`, or `stop`.
A rule trigger indicates potential risk and is not an automatic stop decision. Do not tune parameters, diagnose hardware, execute stop commands, or judge final task performance.

Current Health Event:
{`HEALTH_EVENT`}

Recent Trial History:
{`TRIAL_HISTORY`}

Available Tools:
{`AVAILABLE_TOOLS`}

Review Principles:

1. Assess whether the anomaly persists and whether surrounding training trajectories support the concern or indicate recovery. When available, retrieve current training metrics before recommending `observe` or `stop`.
2. Recommend `stop` only when the evidence supports a severe, persistent anomaly without signs of recovery. When evidence is incomplete or contradictory, recommend `observe` or `continue`.
3. For `observe`, specify the number of additional updates needed for reassessment. During a scheduled follow-up, retrieve updated metrics and compare the observation window with the original event and assessment. A follow-up does not require another rule trigger.

Review Rules and Output:
[Detailed review rules, tool-call requirements, and the JSON output format are omitted.]

Example health assessments. This training-health example comes from a separate tuning run from the earlier Proposal Agent example. Trial IDs below refer to this run. Figure 6 shows the early training trajectories of Trial 09 alongside Trials 07 and 08. At update 5, the change in Trial 09's KL loss exceeded both the absolute and relative thresholds, so the deterministic monitor submitted an alert to the Train Health Agent. Because training reward was still improving and entropy had not collapsed, the Agent returned `watch`/`observe` and requested a reassessment after five more updates:

Example Output of the Train Health Agent: Observe at Step 5 (Abridged)

```
{
  "verdict": "watch",
  "action": "observe",
  "confidence": 0.72,
  "reason_codes": [
    "KL_SUDDEN_CHANGE",
    "REWARD_RISING",
    "ENTROPY_RISING",
```

```
    "EARLY_TRIAL"
  ],
  "evidence": [
    "KL loss rose from 0.00866 to 0.02894 at step 5, exceeding the
    absolute and relative change thresholds.",
    "The reward-window means increased from 0.0926 to 0.1113, with no
    reward drawdown."
  ],
  "counterevidence": [
    "Reward and entropy were rising rather than collapsing.",
    "The trial was only at step 5 of 100; successful reference trials
    had also reached KL loss near 0.023–0.027."
  ],
  "observe_for_updates": 5,
  "reason": "The KL change is not corroborated by reward deterioration
    . Observe five more updates to determine whether KL stabilizes or
    continues to diverge."
}
```


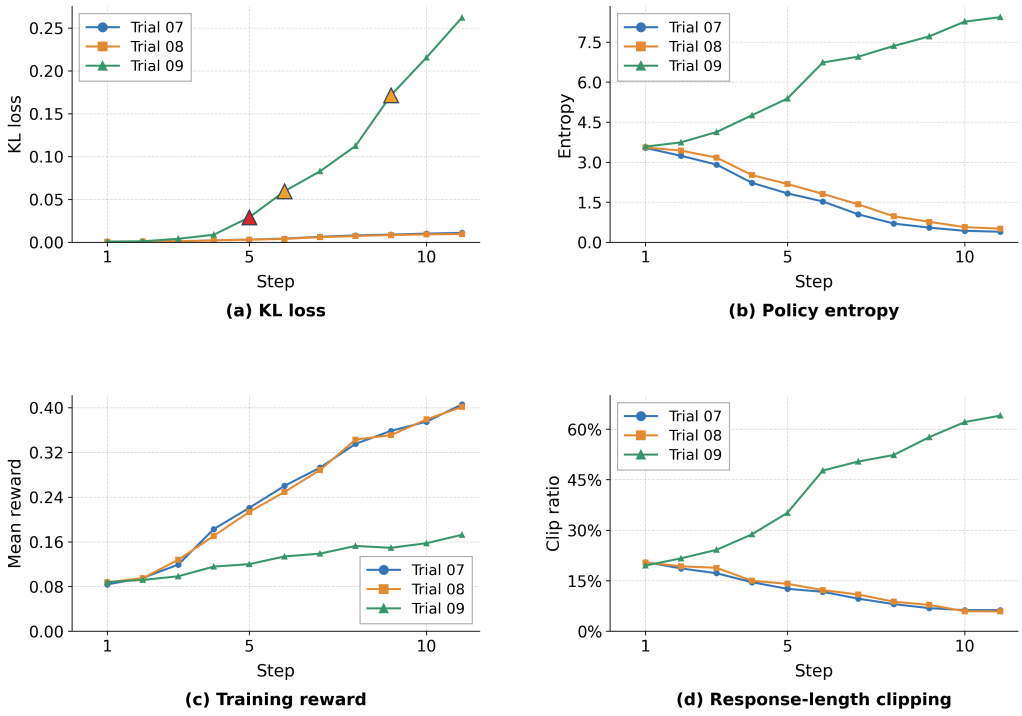

Figure 6: Training-health trajectories for Trials 07–09: (a) KL loss; (b) policy entropy; (c) mean training reward; and (d) response-length clip ratio.

During the observation period, the KL-change rule was satisfied again at updates 6 and 9, as marked in Figure 6(a). The cooldown suppressed new alerts, so neither rule hit caused another Agent call or postponed the scheduled review. At update 10, the observation period expired and the Agent re-assessed the trial without a new rule alert. KL loss had continued to rise, accompanied by increasing entropy and response-length clipping, although reward was still improving. The Agent therefore returned `stop`.

Training continued asynchronously while the Agent generated its response. Consequently, metrics from update 11 were also recorded before the stop action took effect, explaining why the trajectories in Figure 6 extend to update 11.

Example Output of the Train Health Agent: Stop at Step 10 (Abridged)


```
{
  "verdict": "unhealthy",
  "action": "stop",
  "confidence": 0.63,
  "reason_codes": [
    "KL_PERSISTENT_DIVERGENCE",
    "ENTROPY_RISING_ABNORMAL",
    "RESPONSE_LENGTH_DRIFT"
  ],
  "evidence": [
    "KL loss rose from 0.0289 at step 5 to 0.2154 at step 10 without
    stabilizing.",
    "The KL-change rule was satisfied again at steps 6 and 9, but
    cooldown suppressed new alerts.",
    "Policy entropy rose from 5.38 to 8.26, and the response-length
    clip ratio reached 0.62."
  ],
  "counterevidence": [
    "Training reward continued to improve from 0.1199 to 0.1574.",
    "Gradient norms remained bounded, and the trial had completed only
     10 of 100 planned updates."
  ],
  "observe_for_updates": 0,
  "reason": "The scheduled reassessment found persistent KL growth
    accompanied by rising entropy and response-length clipping.
    Despite improving reward, the Agent recommends stopping the trial
    ."
}
```


B.7 Context Supplied to the Agent

For each Agent invocation, the orchestrator fills the prompt with the current tuning stage, parameter permissions, trial evidence, constraints, and available tools. We organize configuration parameters into globally fixed parameters (G), system-tuning parameters (H), and learning-tuning parameters (L). The stage determines which group may be edited and which groups must retain their current values.

Globally fixed parameters. The set G contains model, workload, and training settings that remain fixed throughout both tuning stages within a run. The block below shows values from one prompt instance; file paths and experiment identifiers are redacted.

GLOBALLY FIXED PARAMETERS


```
{
  "algorithm.adv_estimator": "grpo",
  "data.train_files": "***/data/***.parquet",
  "data.val_files": "***/data/***.parquet",
  "data.filter_overlong_prompts": true,
  "data.truncation": "error",
  "actor_rollout_ref.model.path": "***/model/Qwen3-***",
  "actor_rollout_ref.model.use_remove_padding": true,
  "actor_rollout_ref.actor.ppo_mini_batch_size": 256,
 "actor_rollout_ref.actor.megatron.use_distributed_optimizer": true,
  "actor_rollout_ref.actor.megatron.use_remove_padding": true,
  "actor_rollout_ref.actor.megatron.optimizer_offload": true,
```

```
  "actor_rollout_ref.actor.megatron.param_offload": true,
  "actor_rollout_ref.rollout.name": "vllm",
  "actor_rollout_ref.rollout.disable_log_stats": false,
  "algorithm.use_kl_in_reward": false,
  "trainer.critic_warmup": 0,
  "trainer.logger": [ "console", "wandb"],
  "trainer.project_name": "***",
  "trainer.experiment_name": "***",
  "trainer.n_gpus_per_node": 8,
  "trainer.nnodes": 1,
  "trainer.val_before_train": false,
  "trainer.save_freq": -1,
  "trainer.test_freq": 5,
  "trainer.total_epochs": 15
}
```

Current stage. The `CURRENT_STAGE` field indicates whether the current invocation concerns system tuning or learning tuning.

CURRENT_STAGE

```
system_tuning
learning_tuning
```

During system tuning, `EDITABLE_PARAMETERS` permits changes to model parallelism, phase-specific batching, and rollout capacity. These parameters can affect both GPU memory use and end-to-end training throughput. The list specifies which keys may be changed; it does not prescribe their values or an adjustment direction.

EDITABLE_PARAMETERS (system_tuning)

```
[
  "actor_rollout_ref.actor.megatron.sequence_parallel",
  "actor_rollout_ref.actor.megatron.tensor_model_parallel_size",
  "actor_rollout_ref.actor.megatron.pipeline_model_parallel_size",
  "actor_rollout_ref.actor.megatron.virtual_pipeline_model_parallel_size",
  "actor_rollout_ref.actor.megatron.expert_model_parallel_size",
  "actor_rollout_ref.actor.megatron.context_parallel_size",
  "actor_rollout_ref.actor.megatron.expert_tensor_parallel_size",
  "actor_rollout_ref.actor.ppo_max_token_len_per_gpu",
  "actor_rollout_ref.actor.ppo_micro_batch_size_per_gpu",
  "actor_rollout_ref.actor.use_dynamic_bsz",
  "actor_rollout_ref.ref.log_prob_max_token_len_per_gpu",
  "actor_rollout_ref.ref.log_prob_micro_batch_size_per_gpu",
  "actor_rollout_ref.ref.log_prob_use_dynamic_bsz",
  "actor_rollout_ref.rollout.gpu_memory_utilization",
  "actor_rollout_ref.rollout.log_prob_max_token_len_per_gpu",
  "actor_rollout_ref.rollout.log_prob_micro_batch_size_per_gpu",
  "actor_rollout_ref.rollout.log_prob_use_dynamic_bsz",
  "actor_rollout_ref.rollout.max_num_batched_tokens",
  "actor_rollout_ref.rollout.max_num_seqs",
  "actor_rollout_ref.rollout.tensor_model_parallel_size"
]
```

During learning tuning, the editable list covers learning rate, KL and entropy controls. System parameters are held fixed so that candidates can be compared using training trajectories and task evaluation results under the selected system configuration.

EDITABLE_PARAMETERS (learning_tuning)

```
[
  "actor_rollout_ref.actor.entropy_coeff",
```

```
  "actor_rollout_ref.actor.kl_loss_coef",
  "actor_rollout_ref.actor.kl_loss_type",
  "actor_rollout_ref.actor.optim.lr",
  "actor_rollout_ref.actor.optim.lr_warmup_steps",
  "actor_rollout_ref.actor.use_kl_loss",
]
```


Read-only context and proposal constraints. `IMMUTABLE_CONTEXT` provides evidence about the model, hardware, workload, and observed runtime behavior. The Agent may use these facts to assess compatibility and resource risk.

IMMUTABLE_CONTEXT (hardware_tuning example)


```
{
  "model": {
    "model_path": "***/model/Qwen3-***",
    "model_config": {
      "model_type": "qwen3",
      "hidden_size": 1024,
      "num_hidden_layers": 28,
      "num_attention_heads": 16,
      "num_key_value_heads": 8,
      "intermediate_size": 3072,
      "vocab_size": 151936
    },
    "logged_parameter_profile": {
      "most_loaded_shard_parameters": 93979136,
      "total_parameters": null,
      "total_parameters_source": null,
      "complete_tp_pp_coverage": false,
      "reference_topology": {
        "tensor_model_parallel_size": 4,
        "pipeline_model_parallel_size": 2,
        "expert_model_parallel_size": 1,
        "expert_tensor_parallel_size": 4
      }
    },
    "resolved_runtime": {
      "tensor_model_parallel_size": 4,
      "pipeline_model_parallel_size": 2,
      "context_parallel_size": 1,
      "expert_model_parallel_size": 1,
      "expert_tensor_parallel_size": 4,
      "virtual_pipeline_model_parallel_size": null,
      "sequence_parallel": true,
      "recompute_granularity": null,
      "recompute_method": null,
      "recompute_num_layers": null,
      "recompute_modules": "['core_attn']",
      "attention_backend": "<AttnBackend.flash: 1>",
      "num_layers_in_first_pipeline_stage": null,
      "num_layers_in_last_pipeline_stage": null,
      "fp16": false,
      "bf16": true,
      "params_dtype": "torch.bfloat16",
      "num_layers": 28,
      "mtp_num_layers": null,
      "hidden_size": 1024,
      "num_attention_heads": 16,
      "num_query_groups": 8,
      "ffn_hidden_size": 3072,
      "kv_channels": 128,
      "gated_linear_unit": true,
      "num_moe_experts": null,
      "multi_latent_attention": false,
      "moe_shared_expert_intermediate_size": null,
      "moe_layer_freq": 1,
      "moe_ffn_hidden_size": null
    },
    "observed_sequence_length": {
      "point_tokens": 1905.655859375,
```

```
      "upper_tokens": 3031,
      "configured_upper_tokens": 3072,
      "source": "train_log_stable_step_p95_mean_tokens",
      "sampled_steps": 5,
      "mean_tokens_across_sampled_steps": 1860.850859375
    }
  },
  "hardware": {
    "platform": "C550",
    "nnodes": 1,
    "gpus_per_node": 8,
    "world_size": 8,
    "observed_device_memory_mib": 65536,
    "resource_memory_reserve_mib": 4096,
    "throughput_memory_reserve_mib": 8192
  },
  "workload": {
    "algorithm": "grpo",
    "train_dataset": "***/data/***.parquet",
    "evaluation_dataset": "***/data/***.parquet",
    "train_batch_size": 1024,
    "max_prompt_length": 1024,
    "max_response_length": 2048
  },
  "runtime_relationships": {
    "ref_model_parallel_topology": "inherits_actor",
    "entropy_calculation": "training calculate_entropy iff entropy_coeff != 0"
  }
}
```


Compact reference history. The `COMPACT_REFERENCE_HISTORY` field provides selected prior trials with their outcomes and phase-relevant metrics. The two stages, system tuning stage and learning tuning stage, are assembled on demand. A portion is shown below.

SOME METRICS IN COMPACT REFERENCE HISTORY


```
{
  "trial_id": 3,
  "result": "success",
  "updates_completed": 5,
  "metrics": {
    "phase_memory_mib": {
      "rollout": {
        "mean_used_mib": 52178.65,
        "p95_used_mib": 55745.00,
        "max_used_mib": 56435.00
      },
      "actor_log_prob": {
        "mean_used_mib": 9697.23,
        "p95_used_mib": 14857.00,
        "max_used_mib": 14913.00
      },
      "ref_log_prob": {
        "mean_used_mib": 8084.61,
        "p95_used_mib": 9955.00,
        "max_used_mib": 9967.00
      },
      "training": {
        "mean_used_mib": 15938.71,
        "p95_used_mib": 17901.0,
        "max_used_mib": 17901.0
      }
    },
    "phase_duration_s": {
      "rollout": {
        "mean": 255.25,
        "p95": 337.50,
        "max": 381.62
      },
      "actor_log_prob": {
        "mean": 83.53,
```

```
        "p95": 98.01,
        "max": 101.06
      },
      "ref_log_prob": {
        "mean": 86.47,
        "p95": 101.79,
        "max": 105.46
      },
      "training": {
        "mean": 208.90,
        "p95": 243.18,
        "max": 252.64
      }
    },
    "summary": {
      "throughput": {
        "mean": 1828.96,
        "p95": 1836.98,
        "max": 1836.98
      },
      "actor_mfu": {
        "mean": 0.19915,
        "p95": 0.19954,
        "max": 0.19954
      }
    }
  }
}
{
  "windows": [
    {
      "start_step": 3,
      "end_step": 7,
      "sample_count": 5
    }
  ],
  "windows_metrics": {
    "critic/rewards/mean": 0.8227,
    "actor/ppo_kl": 2.01507,
    "actor/kl_loss": 0.06017,
    "actor/entropy": 0.06847,
    "actor/pg_loss": 0.0013,
    "actor/pg_clipfrac": 0.0006,
    "actor/lr": 1e-06
  }
}
```


B.8 Details of Evidence Tools

The system provides nine tools for retrieving parameter knowledge, estimating resource requirements, and inspecting experimental evidence. The tool registry exposes only the tools permitted for each agent and rejects unauthorized calls. Table 5 summarizes the tool functions and access permissions. The Summary Agent synthesizes the supplied run facts without calling tools.

Table 5: Agent tools and their access permissions. P: Proposal; F: Feasibility; D: Diagnosis; H: Train Health; S: Summary. A checkmark (✓) indicates permission to call the tool; a dash (—) indicates no permission.

Tool	Function	P	F	D	H	S
`parameter_understanding`	Retrieves curated verl parameter semantics, memory and throughput effects, interactions, and hard constraints to support informed parameter changes.	✓	✓	✓	—	—
`tuning_strategies`	Retrieves stage-aware guidance on parallelism topology and batching or concurrency settings for measured bottlenecks and hardware failures.	✓	—	—	—	—
`memory_estimator`	Uses an explicitly selected measured reference trial and parameter changes to estimate peak memory and relative changes for rollout, actor log-probability, reference log-probability, and training phases. Reports memory safety; the actual resource gate remains authoritative.	✓	✓	—	—	—
`analyze_rollout_metrics`	Analyzes recorded vLLM metrics for evidence that the memory budget, sequence limit, or batched-token limit constrains rollout. Returns a queue-demand target when the sequence limit is binding; missing metrics remain unknown.	✓	✓	—	—	—
`query_trial_history`	Retrieves one to five reference trials by ID, including stage-specific editable parameters, recorded changes, and compact hardware or stability metrics. Learning-stage results include the MATH validation series.	✓	✓	✓	✓	—
`query_tuning_summaries`	Searches Summary Agent results from earlier runs for useful and ineffective tuning directions. These results provide prior hypotheses; source-run trial IDs cannot serve as current-run references.	✓	—	—	—	—
`read_trial_log_excerpt`	Reads a bounded excerpt from a log already associated with a recorded trial, optionally filtered by a case-insensitive pattern. Arbitrary file paths are not accepted.	—	—	✓	—	—
`read_trial_metrics`	Reads ordered, bounded windows of selected learning-stage metrics from a recorded trial's structured metrics file, automatically including the MATH validation series within the same step range.	✓	—	✓	—	—
`read_current_trial_metrics`	Reads selected metrics from the active trial through the immutable snapshot step supplied by the runner, including MATH validation results. Supports coarse trajectory windows and per-step inspection; another trial ID or file path cannot be supplied.	—	—	—	✓	—