RESHAPING TOOL CALLING POLICIES OF TIME SERIES AGENTS VIA REINFORCED EVALUATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Time series reasoning, which involves understanding temporal patterns to answer complex questions, plays a crucial role in a variety of domains such as healthcare, energy, and climate. Modern approaches have tried using large language model (LLM) based agents with analytical tools to reason over time series. However, these methods would always execute tool calls directly without evaluation, which inevitably leads to many *low-value tools being executed* and *uninformative results being generated*, thus hindering both the reasoning efficiency and accuracy. One seemingly workable solution inspired by LLM and agent studies is to introduce post-training to guide agents for wiser tool calling. However, tuning these massive parameters of LLM-based agents further imposes additional computational burden. To address these problems, in this paper, we explore a novel direction of *evaluating time series reasoning without post-training base agents*. The core idea is to bring an evaluation reward model to assess each tool call using accumulated evidence, whose evaluation can further illuminate exploration of time series agents and reshape their action strategies. Naturally, we reformulate the iterative evaluation and action-reshaping process into a Markov decision problem that maximizes the expected reward composed of tool costs and task performance. To optimize the objective, we propose a novel time series agentic framework, *TimesEval*, which uses value-based reinforcement learning to learn evaluation policies for reshaping agent actions. To estimate rewards for each evaluation step, we propose episode relative advantage learning, which measures the expected return difference among decisions as the advantage for fast convergence of action reshaping. Moreover, to approach the global optimal policy, we utilize evolving policy optimization, which approximates policy iteration with on-policy state sampling to progressively improve evaluation reward model. Notably, the key difference is TimesEval trains a lightweight evaluator through reinforcement learning to reshape agent actions instead of post-training the agent itself. Finally, we conduct extensive experiments on 15 datasets and across 4 time series tasks, demonstrating that TimesEval consistently achieves state-of-the-art performance, with an average improvement of more than 19% over the strongest baseline and 36% lower tool cost.

1 INTRODUCTION

Time series reasoning, which focuses on analyzing time series data to answer complex questions and support tasks such as forecasting (Wang et al., 2026), classification (Liu et al., 2025b), and analytical question answering (Kong et al., 2025), has become increasingly important across diverse application domains, including healthcare (Li et al., 2026b), energy (Jiang et al., 2026), and climate (Huang et al., 2026). With the recent advancement of Agentic AI (Jin et al., 2026), methods powered by large language model (LLM) based agents have been proposed for time series tasks by taking actions, observing their outcomes, and planning the next steps (Liu et al., 2025a; Li et al., 2026c). By invoking external tools, agents can perform precise numerical analysis, such as statistical characterization (Wu et al., 2026), and then incorporate their results into reasoning process.

While these agents have achieved promising performance in time series reasoning tasks, their direct execution of tool calls without evaluation often leads to ineffective tool invocations and generates low-information results for agents, which brings an extra burden to the reasoning process and degrades the performance (Wang et al., 2025; Chen et al., 2025b; Qian et al., 2025). For example, the

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.

agent in Figure 1 (left) makes ineffective tool invocations that analyze the trend again after obtaining the result of ComputeSlope, while Figure 1 (right) shows that executing more tools generates additional results and lowers the accuracy. These observations suggest that time series agents require wiser tool calling to ensure both the efficiency and performance of reasoning (Jin et al., 2026).

Following current studies on LLM-based agents, a natural solution is to post-train time series agents for effective tool use (Qian et al., 2026; Feng et al., 2025). These methods primarily follow two lines: one applies supervised fine-tuning to time series agents on trajectories to learn a strategic tool-use policy (Wu et al., 2026), while the other adopts reinforcement learning to optimize time series reasoning models with outcome and process rewards (Lu et al., 2026; Zhou et al., 2026; Liu et al., 2025c). However, these post-training methods often need to tune LLM-based agents with large-scale training data and billion parameter optimization, which further increases the computational burden.

To address these problems, we explore a new perspective of *evaluating time series reasoning without post-training base agents*. The core principle is to introduce an evaluation reward model to assess each tool call based on the accumulated evidence before execution, whose evaluation can guide the exploration of time series agents and reshape their action policies. Accordingly, we reformulate the interleaved process of evaluating tool calls and reshaping agent actions as a Markov decision process with the objective of maximizing the expected reward balancing performance and tool costs. To optimize this objective, we propose a time series agentic framework, *TimesEval*, which learns evaluation policies through value-based reinforcement learning and applies them to reshape agent actions during reasoning. To estimate the rewards of the evaluation step, we propose episode relative advantage learning, which rolls out different decisions from the same state and measures their expected return differences as advantages. These advantages further serve as process rewards, which enable fast convergence and effectively reshape agent actions. Furthermore, to approach the global optimal evaluation policy, we utilize evolving policy optimization, which iteratively retrains the evaluation reward model with on-policy state sampling. In summary, our contributions are:

- We explore a new direction of *evaluating time series reasoning without post-training base agents*. Unlike existing retraining methods that tune the massive parameters of agents, we reformulate the interleaved process of evaluating tool calls and reshaping agent actions into a Markov decision process that maximizes the expected reward balancing task performance and tool costs.
- We propose *TimesEval*, a time series agentic framework that learns evaluation policies through value-based reinforcement learning and applies them to reshape agent actions. To estimate process rewards, we propose episode relative advantage learning, which uses the expected return difference among decisions as the advantage for fast convergence of action reshaping. To approach the global optimal policy, we further utilize evolving policy optimization, which progressively improves the evaluation reward model through policy iteration with on-policy state sampling.
- We conduct experiments on TSRBench (Yu et al., 2026), covering 15 datasets across 4 tasks with four backbone models, which demonstrate that TimesEval outperforms existing tool-use methods. TimesEval achieves a 19% relative improvement in overall performance over the strongest baseline, while reducing the interaction cost by 36% compared with executing all tools.

2 Related Works

Time Series Reasoning. Time series reasoning focuses on understanding temporal structures and drawing conclusions from sequential numerical observations (Chang et al., 2025). It has become an increasingly important capability for large language models (LLMs) (Luo et al., 2025; Chang et al., 2025). Existing LLM-based studies cover diverse tasks, including numerical forecasting (Jin et al., 2024), trend classification (Zhou et al., 2025), and analytical question answering (Yugeswardeenoo et al., 2024), with applications in domains such as weather (Li et al., 2025a) and finance (Yu et al., 2024). However, these methods often struggle when the evidence required for reasoning is spread across numerical temporal dynamics and unstructured textual information. To incorporate both sources of evidence, recent multimodal time series LLMs jointly process numerical sequences and contextual text, including news articles (Wang et al., 2024), market commentary (Kawarada et al., 2024), and weather reports (Li et al., 2025a). By combining these modalities, such models support more complex temporal reasoning problems that cannot be addressed using numerical observations or textual context alone (Guan et al., 2025; Jin et al., 2024; Zhong et al., 2025).

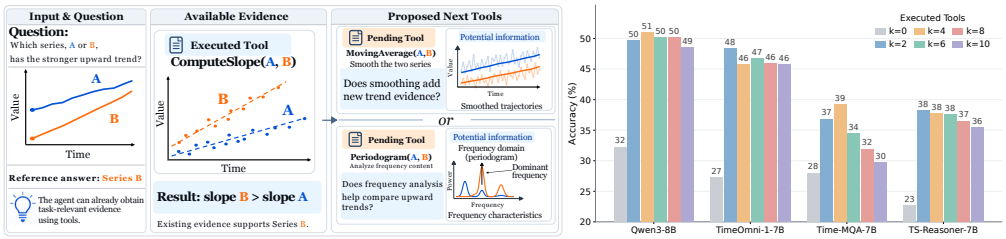


Figure 1: Visualizations of tool execution in time series reasoning. **Left:** Multiple tools can describe the same property, but they provide different amounts of new evidence. **Right:** Executing tools improves accuracy over using no tools ($k = 0$), while executing more tools can degrade performance.

Agentic Tool Learning. Existing approaches to tool-augmented LLM agents follow two complementary directions. The first improves inference-time tool use while keeping the base policy fixed: ReAct alternates reasoning and acting (Yao et al., 2022), ToolChain (Zhuang et al., 2024), LATS (Zhou et al., 2023), and ToolTree (Yang et al., 2026) search over action trajectories, GTool models tool dependencies as graphs (Chen et al., 2025a), DreamPhase evaluates imagined futures (Hamidi et al., 2026), and Planning without Search learns a lightweight goal-conditioned value function to guide a fixed LLM (Hong et al., 2026). The second directly optimizes the tool-use policy through supervised, preference, or reinforcement learning: ToolLLM, TRICE, and CoRT learn from tool-use trajectories, execution feedback, or code-integrated reasoning data (Qin et al., 2024; Qiao et al., 2024; Li et al., 2026a). ToolRL designs tool-specific rewards (Qian et al., 2026), ReTool learns to invoke tools through outcome-driven RL (Feng et al., 2025), Tool-Light balances insufficient and excessive tool use through preference learning (Chen et al., 2025b), and Tree-GRPO and ASearcher improve step-level advantage estimation and scale end-to-end RL (Ji et al., 2025; Gao et al., 2026).

3 Preliminaries

This paper studies *time series reasoning* (Yu et al., 2026; Zhang et al., 2025): given a time series x and a question q, an LLM-based agent π_θ produces an answer $\hat{y}$ by interacting with a tool set $\mathcal{T} = \{T_1, \ldots, T_M\}$. At step t, the agent samples an action $a_t \sim \pi_\theta(\cdot \mid q, x, h_t)$ from $\mathcal{C} \cup \mathcal{Y}$, where each tool call $c \in \mathcal{C}$ specifies a tool $\mathrm{tool}(c) \in \mathcal{T}$ with arguments, $\mathcal{Y}$ is the answer space, and h_t is the history of executed calls and evidence with $h_1 = \emptyset$. A tool call $c_t = a_t$ returns the observation $o_t = c_t(x)$ and extends the history to $h_{t+1} = h_t \,\|\, (c_t, o_t)$, while an answer $a_t \in \mathcal{Y}$ terminates with $\hat{y} = a_t$. The agent answers within a budget $T_{\max}$, so the termination time is $T = \min\{t \le T_{\max} : a_t \in \mathcal{Y}\}$ and the trajectory $\tau = \big((c_1, o_1), \ldots, (c_{T-1}, o_{T-1}), \hat{y}\big)$ has policy likelihood:

$$\pi_\theta(\tau \mid q, x) = \prod_{t=1}^{T} \pi_\theta\big(a_t \mid q, x, h_t\big). \tag{1}$$

For cost accounting, we also define the execution trace ξ, which records every invocation of the base agent, including proposed actions that are not executed, together with each executed tool call.

4 TimesEval

In this section, we first formulate tool execution control as a Markov decision process (Section 4.1). We then present our value-based reinforcement learning framework (Section 4.2), the evaluator trained with episode relative advantages (Section 4.3), and the evolving policy optimization (Section 4.4). As illustrated in Figure 2, TimesEval instantiates this framework.

4.1 Reformulating Time Series Reasoning as a Markov Decision Process

Existing time series agents learn to use tools through reinforcement learning, with task performance as reward (Wu et al., 2026). However, such an objective does not consider computational cost, which may lead to unnecessary computation on simpler problems (Hamrick et al., 2017). Moreover, deciding whether to execute a tool call requires considering current evidence and its effects on

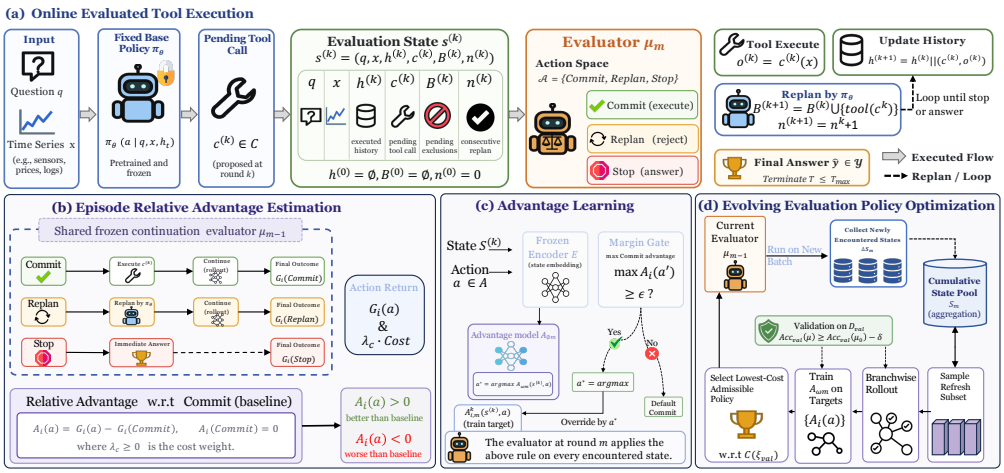


Figure 2: The framework overview of TimesEval. **(a)** During online reasoning, the evaluation reward model reshapes each pending agent action. **(b)–(d)** illustrate the training of the evaluation reward model: (b) Relative advantage estimation from same-state rollouts. (c) Training of the evaluation reward model. (d) Evolving policy optimization with on-policy state sampling.

subsequent tool use (Hay et al., 2014), so evaluating tool calls and reshaping agent actions naturally form a Markov decision process (MDP) (Callaway et al., 2017).

In the conventional tool-use formulation, the base policy π_θ selects $a_t \in \mathcal{C} \cup \mathcal{Y}$ at state $s_t = (q, x, h_t)$, where $\mathcal{C}$ and $\mathcal{Y}$ denote tool calls and answers, respectively. A tool call appends its returned observation to h_t, whereas an answer terminates the trajectory. The policy maximizes the expected reward for the task $\max_\theta \mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{T} r(s_t, a_t)\right]$, where r denotes the reward based on the final answer. Thus, trajectories with the same task return are valued equally even when their interaction costs differ.

More importantly, an agent can solve tasks with tools but may struggle to control the computational cost of tool use (Wang et al., 2025). We therefore seek to preserve task performance while reducing unnecessary tool costs. Motivated by constrained reinforcement learning (Achiam et al., 2017), we treat task performance as a constraint and minimize tool cost. To implement this control, we introduce an evaluation policy on the fixed agent, with interaction cost as reward.

To describe this process, we model this process as an MDP (Puterman, 2014) $\mathcal{M}_{\text{eval}} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \rho, H_{\max})$. We instantiate the state space $\mathcal{S}$, action space $\mathcal{A}$, and transition dynamics $\mathcal{P}$ in Section 4.3. The reward $\mathcal{R}$ corresponds to negative interaction cost in Equation 2, while ρ and $H_{\max}$ denote the initial-state distribution and the maximum number of evaluation rounds.

Let $\mathcal{M}$ denote the class of runtime evaluation policies, and let $\xi_{\pi_\theta,\mu}$ and ξ_{π_θ} denote complete execution traces with and without evaluation, respectively. For a trace ξ, the weighted interaction cost is $C(\xi) = N_{\text{tool}}(\xi) + \lambda_{\text{LLM}} N_{\text{LLM}}(\xi)$, where $N_{\text{tool}}(\xi)$ and $N_{\text{LLM}}(\xi)$ count executed tool calls and LLM invocations, respectively. Over $(q, x, y) \sim \mathcal{D}$, we define $\text{Acc}(\mu) = \mathbb{E}_{\mathcal{D},\pi_\theta,\mu}[\mathbb{I}[\hat{y}(\xi_{\pi_\theta,\mu}) = y]]$. We rewrite the reward as $R(\xi) = -C(\xi)$ to optimize tool execution efficiency. Let $\mu_0 = \text{Always-Commit}$ denote the reference policy that executes each tool. We propose the following objective, where δ denotes the allowed accuracy degradation:

$$\max_{\mu \in \mathcal{M}} \mathbb{E}_{\mathcal{D},\pi_\theta,\mu}[-C(\xi_{\pi_\theta,\mu})] \quad \text{s.t.} \quad \text{Acc}(\mu) \ge \text{Acc}(\mu_0) - \delta. \tag{2}$$

4.2 Learning to Evaluate Tool Execution for Time Series Reasoning

To optimize the objective in Section 4.1, we aim to learn an evaluation policy that controls tool execution throughout the time series reasoning process. However, the optimal evaluation is difficult to determine, as it requires considering not only the value of directly executing tool but also how its results shape agent's subsequent reasoning (Hamrick et al., 2017). We therefore propose a value-based reinforcement learning (RL) framework that trains an evaluation reward model, *evaluator*, to

learn the relative values of evaluation decisions from experience and improve the evaluation policy for tool execution while keeping the base agent fixed.

The objective of RL is to find a policy π that maximizes the expected discounted return in the MDP formulated in Section 4.1. The action-value function $Q^\pi(s,a)$ represents the expected discounted return of taking action a in state s and following π. It satisfies the Bellman expectation equation:

$$Q^\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)} \left[V^\pi(s') \right].$$

A policy can be derived from Q^π by maximizing over actions at each state. In our framework, we estimate action values from episode returns, where each episode begins with a feasible decision at a sampled state and follows the current evaluation policy μ. We compare the value of each decision with that of default execution, Commit, and maximize the relative action value to obtain a policy μ':

$$\mu'(s) \in \arg\max_{a \in \mathcal{A}(s)} Q^\mu(s,a) = \arg\max_{a \in \mathcal{A}(s)} \left[Q^\mu(s,a) - Q^\mu(s, \mathsf{Commit}) \right].$$

At each evaluation round, the evaluation state $s \in \mathcal{S}$ captures the task, accumulated evidence, and pending tool proposal. Given this state, the evaluator A_ω predicts action values, and the evaluation policy $\mu(\cdot \mid s)$ selects a feasible decision $a \in \mathcal{A}(s)$. This decision determines whether to execute the tool, request replanning, or terminate tool execution. The process then either transitions to a new evaluation state $s' \sim \mathcal{P}(\cdot \mid s, a)$ or terminates with a final answer from the base agent. From the perspective of the evaluation policy, the base agent and the tools together form the environment.

4.3 Training the Evaluation Reward Model with Relative Advantages

To estimate step rewards for evaluation, we train an evaluation reward model, evaluator, to measure return difference among decisions. Prior work trains agents to use tools through RL using returns from complete trajectories (Wang et al., 2025; Li et al., 2025b). However, these returns reflect the combined effects of multiple actions, making the step-level credit assignment challenging. Different from these methods, our evaluator learns evaluation decisions on the actions proposed by the base agent, so that each evaluation decision can be measured directly. Motivated by relative advantage estimation in RL (Sutton et al., 1998), we propose episode relative advantage learning, which measures returns from decisions to train the evaluator for effectively reshaping agent actions.

Specifically, we instantiate the framework in Section 4.2 for evaluated time series reasoning by specifying the states, decisions, and evaluator training. (i) *State representation* captures the evidence and the pending agent action; (ii) *Evaluation decision space* defines evaluation decisions and their effects on subsequent states; (iii) *Episode relative advantage learning* measures expected return difference of decisions from the same state to train the evaluator, which further guides agent actions.

Evaluation State. Following prior work (Callaway et al., 2017), we encode the task, accumulated evidence, and pending agent action in the evaluation state, which serves as the input of the evaluation reward model. At round k, we define $s^{(k)} \in \mathcal{S}$ as:

$$s^{(k)} = \left(q, x, h^{(k)}, c^{(k)}, B^{(k)}, n^{(k)} \right), \tag{3}$$

where q and x denote the question and input time series, $h^{(k)}$ records executed tool calls and accumulated evidence, and $c^{(k)}$ is the pending agent action. The set $B^{(k)} \subseteq \mathcal{T}$ contains tools excluded under the current evidence, while $n^{(k)}$ counts replanning steps.

Evaluation Decision Space. To control agent actions based on the current state, the evaluation policy needs a decision space that supports reshaping the agent actions. Existing work integrates tool generation and execution within the agent's reasoning process (Li et al., 2025b), whereas we decouple them and evaluate each agent action before it is executed. We therefore propose three evaluation decisions and their transition rules to continue, redirect, or terminate tool use.

(1) Commit. We execute the agent action $c^{(k)}$ to obtain $o^{(k)} = c^{(k)}(x)$ and update $h^{(k+1)} = h^{(k)} \,\|\, (c^{(k)}, o^{(k)})$. The exclusion set and replanning count are reset: $B^{(k+1)} = \emptyset$ and $n^{(k+1)} = 0$. Given the updated history, the base agent either proposes $c^{(k+1)}$ or terminates with a final answer.

(2) Replan. The pending agent action is rejected while the accumulated evidence remains unchanged, so $h^{(k+1)} = h^{(k)}$. To avoid repeatedly proposing the same tool under unchanged evidence,

we update the exclusion set as $B^{(k+1)} = B^{(k)} \cup \{\mathrm{tool}(c^{(k)})\}$ and increment the consecutive replanning count: $n^{(k+1)} = n^{(k)} + 1$. The base agent policy is then asked to propose a new action $c^{(k+1)}$ using the current evidence and the updated exclusion set.

(3) Stop. The base agent policy generates the final answer $\hat{y}$ from the current evidence without further actions. The process then transitions to a terminal state.

Episode Relative Advantage Learning. With the states and decisions defined, we need to estimate process rewards for each decision. Existing work on time series agents assigns an outcome reward to the trajectory, which makes fine-grained credit assignment challenging (Guan et al., 2025). Following relative advantage estimation in RL (Schulman et al., 2015), we measure the return difference among decisions in the decision space as step-level supervision. These relative advantages serve as process rewards to train the evaluation reward model and further guide agent actions.

At round m, each state $s_i^{(k)}$ is restored as the common initial state of a group of episodes. For each feasible decision $a \in \mathcal{A}_i^{(k)}$, we construct one episode that takes a as its initial decision and, if non-terminal, follows the same frozen evaluation policy μ_{m-1} until termination, yielding the execution trace $\xi_{i,k}^{a,m}$. We evaluate these episodes using final-answer correctness and normalized interaction cost. Since these episodes share the same initial state and policy, we propose a relative advantage target using the episode initialized with COMMIT as the reference:

$$G_{i,m}^{(k)}(a) = \mathbb{I}\Big[\hat{y}\Big(\xi_{i,k}^{a,m}\Big) = y_i\Big] - \lambda_c \frac{C\Big(\xi_{i,k}^{a,m}\Big)}{C_{\max}}, \qquad A_{i,m}^{(k)}(a) = G_{i,m}^{(k)}(a) - G_{i,m}^{(k)}(\mathsf{Commit}). \quad (4)$$

Here, $G_{i,m}^{(k)}(a)$ is the episode return and $A_{i,m}^{(k)}(a)$ is its advantage relative to COMMIT. By construction, $A_{i,m}^{(k)}(\mathsf{Commit}) = 0$, and a positive advantage indicates a better episode outcome than COMMIT under the same evaluation policy. Here, $C_{\max} > 0$ is a fixed upper bound on interaction cost, determined by the maximum number of tool calls and the replan limit, as detailed in Appendix C.

To generalize these relative-advantage targets beyond the evaluated states, we train a lightweight evaluation reward model A_{ω_m} using a frozen state encoder E. The evaluator predicts relative advantages from the current evaluation state. We propose the following advantage objective to train the evaluator using episode-level relative-advantage targets:

$$\mathcal{L}_A(\omega_m) = \frac{1}{|\mathcal{B}_m|} \sum_{s_i^{(k)} \in \mathcal{B}_m} \frac{1}{|\mathcal{I}_i^{(k)}|} \sum_{a \in \mathcal{I}_i^{(k)}} \Big(A_{\omega_m}\Big(E(s_i^{(k)}), a\Big) - A_{i,m}^{(k)}(a)\Big)^2, \quad (5)$$

where $\mathcal{B}_m \subseteq \mathcal{S}_m$ contains the training states sampled from the accumulated state pool and $\mathcal{I}_i^{(k)}$ contains their feasible alternatives to COMMIT. Each trained evaluation reward model induces a policy selecting the feasible decision with the highest predicted relative advantage.

4.4 EVOLVING EVALUATION POLICY OPTIMIZATION

To approach the globally optimal policy, we aim to learn an evaluation policy that is effective throughout the reasoning process. The relative advantages above compare actions under a fixed evaluation policy. However, step-level evaluations may be insufficient to learn an effective evaluation policy for the entire time series reasoning process (Young & Sutton, 2020). Motivated by policy iteration in RL (Winnicki & Srikant, 2023), we propose an iterative optimization procedure for the evaluation policy. Each round updates the evaluator with new states to guide policy improvement.

Starting from μ_0, we optimize the evaluation policy over rounds $m = 1, 2, \ldots$. At round m, the current evaluation policy μ_{m-1} collects newly visited states $\Delta\mathcal{S}_m$, which are added to the accumulated state pool. We then sample states from this pool and refresh their episode relative advantages under μ_{m-1} for different cost weights λ_c. We train evaluators using these targets and construct candidate evaluation policies from different training checkpoints for each cost weight. Let $\mathcal{C}_m$ denote the candidates that satisfy the accuracy requirement. We propose the following update rule for the state pool and evaluation policy, subject to accuracy preservation:

$$\mathcal{S}_m = \mathcal{S}_{m-1} \cup \Delta\mathcal{S}_m, \qquad \mu_m \in \arg\max_{\mu \in \mathcal{C}_m} \widehat{\mathbb{E}}\left[-C(\xi_{\pi_\theta,\mu})\right] \quad \text{s.t.} \quad \widehat{\mathrm{Acc}}(\mu) \geq \widehat{\mathrm{Acc}}(\mu_0) - \delta. \quad (6)$$

Table 1: Main results on TSRBench across 15 datasets in 4 tasks. The best results are in **bold**, and the second-best results are underlined. Full task descriptions are provided in Appendix B.

Model	Method	Perception				Reasoning							Prediction		Decision		Overall
		PR	NU	AD	CA	ER	CD	AR	TR	NR	DR	IR	TSF	EP	QualDM	QuantDM	
Textual Time Series as Input																	
Qwen3-8B	Baseline	51.8	56.3	48.8	56.6	14.0	29.3	57.3	23.1	36.5	23.6	33.0	46.7	48.9	34.9	28.7	39.3
	+ ReAct	57.4	68.2	40.0	70.4	22.1	67.1	73.6	28.7	37.0	59.0	37.5	44.2	38.4	36.6	66.3	49.8
	+ ToolTree	54.4	68.2	46.0	65.3	**24.6**	**68.8**	74.4	28.7	36.1	76.0	36.1	41.4	67.5	35.8	64.6	52.5
	+ LATS	57.8	62.1	48.0	67.3	18.2	55.8	76.0	32.6	35.8	68.5	34.7	45.2	38.8	37.0	63.8	49.4
	+ Ours	**58.4**	**70.1**	**49.2**	**70.7**	24.0	68.3	**76.7**	**34.6**	**40.0**	**90.0**	**38.0**	**47.7**	**69.0**	**37.3**	**67.0**	**56.1**
TimeOmni-1-7B	Baseline	55.0	59.8	41.9	64.6	28.0	35.3	46.7	24.4	31.5	22.8	34.0	30.0	49.4	34.0	30.3	39.2
	+ ReAct	56.4	69.7	35.0	61.2	19.3	36.7	56.2	21.7	28.8	61.0	26.4	28.5	48.4	34.0	64.6	43.2
	+ ToolTree	55.1	72.7	37.0	62.2	24.9	40.8	52.9	22.5	24.4	71.5	33.3	26.1	45.3	36.2	64.2	44.6
	+ LATS	55.4	66.3	42.0	64.3	24.6	55.8	**68.6**	24.0	26.3	67.8	27.8	29.0	58.5	33.2	59.3	46.9
	+ Ours	**57.0**	**76.7**	**47.0**	**65.2**	**29.5**	**56.2**	68.0	**28.9**	**37.5**	**74.9**	**39.2**	**33.8**	**66.4**	**39.1**	**65.4**	**52.3**
Time-MQA-7B	Baseline	46.3	59.1	42.0	63.3	28.8	25.0	47.9	24.8	28.8	33.7	15.3	25.6	24.6	35.9	26.8	35.2
	+ ReAct	46.3	72.7	35.0	65.3	25.3	57.9	48.8	33.3	30.4	**86.3**	23.6	25.2	51.2	33.2	58.4	46.2
	+ ToolTree	44.3	69.7	42.0	62.2	34.0	59.2	56.2	20.2	26.6	83.9	23.6	26.6	50.9	35.9	57.6	46.2
	+ LATS	46.1	65.1	38.0	64.3	28.1	35.8	38.0	23.3	28.8	28.3	22.2	26.3	33.9	31.3	53.9	37.6
	+ Ours	**48.6**	**78.8**	**43.0**	**71.4**	**34.9**	**59.8**	**58.1**	**37.9**	**36.3**	84.5	**28.1**	**28.2**	**54.2**	**38.3**	**59.8**	**50.8**
TS-Reasoner-7B	Baseline	44.6	54.5	43.0	43.9	14.0	9.2	19.8	13.2	20.9	28.3	22.2	13.6	75.1	32.0	27.6	30.8
	+ ReAct	47.6	63.6	35.0	52.0	11.6	18.8	19.2	16.1	31.0	79.6	36.1	15.3	79.2	35.5	49.8	39.4
	+ ToolTree	48.0	65.1	38.0	58.2	11.2	19.2	19.5	16.3	**31.6**	81.6	37.2	13.6	78.9	37.0	51.4	40.5
	+ LATS	44.6	54.5	43.0	43.9	14.0	9.2	18.8	13.2	20.9	25.9	22.2	16.7	75.1	39.2	27.6	31.3
	+ Ours	**59.3**	**66.7**	**55.0**	**65.0**	**16.0**	**41.7**	**31.2**	**24.0**	31.0	**97.6**	**37.5**	**19.1**	**79.6**	**45.5**	**54.8**	**48.3**

Here, $\widehat{\mathrm{Acc}}$ denotes accuracy on validation set. This update retains previously collected states and adds states visited by the current policy, while refreshing relative-advantage targets for sampled states under the current evaluation policy. The selected policy then guides state collection and serves as the frozen evaluation policy for relative-advantage target estimation in the next round.

5 Experiments

5.1 Experimental Setup

Benchmark. We conduct experiments on TSRBench (Yu et al., 2026), a time series reasoning benchmark containing 4,125 problems from 14 domains and 15 tasks organized into four categories: Perception (e.g., PR: Pattern Recognition and NU: Noise Understanding), Reasoning (e.g., ER: Etiological Reasoning and CD: Causal Discovery), Prediction (TSF: Time Series Forecasting and EP: Event Prediction), and Decision-Making (QualDM: Qualitative Decision-Making and QuantDM: Quantitative Decision-Making). For compactness, Table 1 uses abbreviations for the 15 tasks; their full names and detailed descriptions are provided in the supplementary material.

Models. We evaluate the general-purpose Qwen3-8B model (Team, 2025) and three time series oriented models: TimeOmni-1-7B (Guan et al., 2025), which improves unified time series reasoning through supervised fine-tuning and reinforcement learning; Time-MQA-7B (Kong et al., 2025), which formulates diverse time series tasks as natural-language question-answering problems and trains LLMs on the resulting question-answering corpus; and TS-Reasoner-7B (Yu et al., 2025), which connects a time series foundation model to an LLM through a trainable adapter.

Baseline Methods. The baseline methods are designed to improve multi-step reasoning and tool use without additional training. ReAct (Yao et al., 2022) alternates between reasoning steps and task-specific actions, allowing the model to update its plan based on environmental feedback. Building on this framework, LATS (Zhou et al., 2023) explores multiple reasoning–action trajectories using Monte Carlo tree search, guided by language-model-based value estimates and self-reflection. Finally, ToolTree (Yang et al., 2026) focuses on efficient multi-tool planning by combining pre- and post-execution evaluation with bidirectional pruning.

5.2 Main Results

Table 1 demonstrates the effectiveness of our method on TSRBench. Without tools, baseline models achieve limited overall performance. ReAct, ToolTree, and LATS improve upon these baselines across all four backbones, highlighting the value of analytical tools for time series reasoning. Our method further considers whether a pending tool provides useful information given the available evidence by evaluating each tool call before execution. It achieves 19.3% relative improvement in

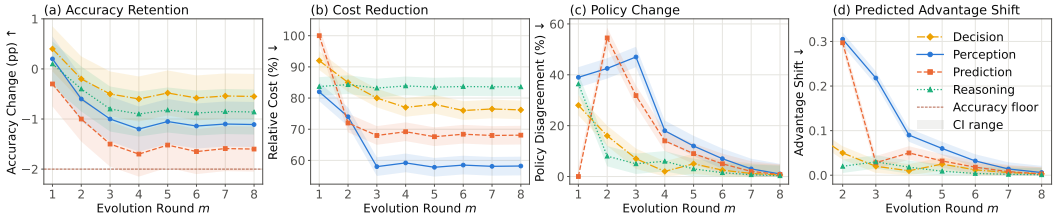


Figure 3: Evolution dynamics of the evaluation policy on the validation set.

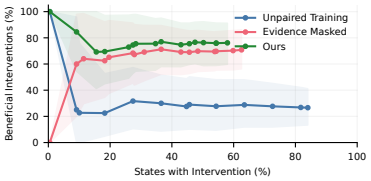


Figure 4: Quality and frequency of evaluation decisions.

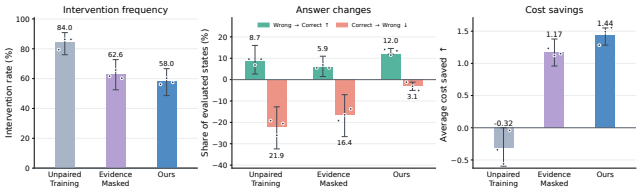


Figure 5: Impact of evaluation decisions on performance and computational cost.

overall performance over ToolTree, the strongest competing method on TS-Reasoner-7B. Within Reasoning task, our method improves the category average across all four backbones, relative gains ranging from 7.8% to 28.5% over the strongest competitor for each backbone. Improvements also extend to Noise Understanding (NU), with relative gains from 2.5% to 8.4% across all four backbones, demonstrating benefits on tasks of interpreting noise in temporal observations.

5.3 Further Analysis

Ablation Study. We assess each component through ablations in Table 2, with implementation details in Appendix A.3. Compared with Always-Commit, our method achieves higher accuracy while reducing the interaction cost by 14.6%. For training supervision, Unpaired Training, which learns from single-branch returns, underperforms Always-Commit with 2.1% cost reduction, indicating that comparing actions from the same state is essential. For evaluator input, Evidence Masked, which removes previous tool outputs from the evaluator, leads to the lowest accuracy, suggesting that the evaluator relies on the obtained evidence to judge each tool call. For policy evolution, Fixed Behavior reduces the cost but violates the accuracy constraint, while w/o Policy Evolution barely reduces the cost, showing that collecting states under the current policy is necessary. These results show that iteratively collecting states under the current policy is necessary to improve performance.

Table 2: Ablation study on Qwen3-8B. Cost denotes the average weighted cost, and Constraint indicates whether the accuracy constraint in Eq. 2 is satisfied. The best results are in **bold**.

Variant	Overall Acc. ↑	Cost ↓	Acc. Constraint
Always-Commit	51.10	3.90	Yes
Unpaired Training	46.86	3.82	No
Evidence Masked	45.33	3.68	No
Fixed Behavior	47.85	3.42	No
w/o Policy Evolution	50.00	3.89	Yes
w/o Accuracy Constraint	48.18	**3.33**	No
w/o Cost-aware Return	49.00	3.93	No
Ours	**55.50**	**3.33**	Yes

Policy Evolution Analysis. In Figure 3, we illustrate the evolution of the policy on the validation set. Panels (a) and (b) show that most cost reductions occur within the first three rounds, while accuracy stays above the floor. Policy disagreement between consecutive rounds peaks at approximately 55% on Prediction and 47% on Perception before approaching zero in panel (c). The change in predicted advantages decreases from approximately 0.3 to nearly zero in panel (d). These trends suggest that the policy stabilizes after a few rounds, with the largest updates coinciding with substantial cost reductions. Cost savings range from approximately 42% on Perception to 16% on Reasoning, indicating different opportunities for reducing tool use across tasks.

Decision Effectiveness. In Figure 5, we analyze the evaluation decisions of different evaluators to examine their effects on reasoning performance and computational cost. Our method selects Replan

or Stop instead of Commit in 58.0% of evaluated states, compared with 84.0% for Unpaired Training and 62.6% for Evidence Masked. Among these decisions, our method changes 12.0% of answers from wrong to correct and only 3.1% from correct to wrong. Both ablations show the opposite pattern, with more answers becoming wrong than correct. Our method also achieves the largest cost savings, while Unpaired Training increases cost through frequent replanning. These results suggest that making more decisions to avoid the proposed tool call does not necessarily improve efficiency. Comparing actions from the same state and incorporating accumulated evidence help reduce unnecessary tool use while limiting harmful changes to the final answer.

Advantage Estimation Quality. In Figure 4, we examine whether decisions with higher predicted advantages are more likely to be beneficial. We rank states where the evaluator selects Replan or Stop by predicted advantage, from highest to lowest. We then progressively include more states and measure the fraction of beneficial decisions among them. Our method keeps about 75% of the decisions beneficial as the proportion increases to 58%. In contrast, Unpaired Training drops to about 25% at an early stage and remains low up to 84% of the states, while the most confident decisions of Evidence Masked are harmful at the beginning. These results show that our method ranks evaluation decisions reliably. Moreover, comparing actions from the same state is essential for advantage estimation, and the obtained evidence is necessary to identify beneficial decisions.

5.3.1 Comparison with Reinforcement Learning Trained Time Series Models

Recent approaches use RL to improve the reasoning capabilities of the base time series model (Lu et al., 2026). Our framework instead learns to control tool execution while keeping the base model fixed. To examine whether learned execution control can achieve competitive reasoning performance at lower computational cost, we compare our framework with these RL-trained models on the same benchmark. Specifically, PATRA uses GRPO with task and format rewards to improve reasoning across different tasks (Lu et al., 2026). Time-R1 uses GRPO with progressive rewards to develop temporal reasoning and event timing prediction capabilities (Liu et al., 2025c). VeriTime applies GRPO to previously unsolved examples, combining outcome and process rewards to improve answer correctness and intermediate reasoning (Zhou et al., 2026).

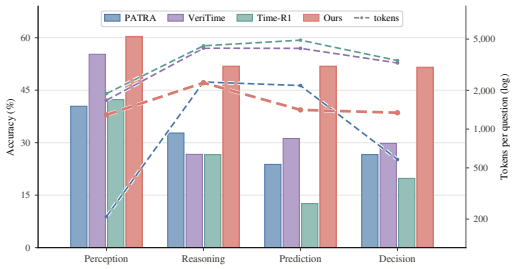


Figure 6: Performance compared with RL-trained models. More results provided in Appendix A.2

As shown in Figure 6, our method achieves the highest accuracy across all four task categories. VeriTime performs competitively on Perception, while our method shows a clearer advantage on the other three categories. Across these three categories, our method also uses fewer tokens per question on average than VeriTime and Time-R1. PATRA uses fewer tokens on two tasks, but its accuracy remains below 41% across all categories. Moreover, unlike VeriTime and Time-R1, whose token usage rises sharply on Reasoning and Prediction, our method keeps a stable token cost across categories by rejecting low-value tool calls before execution. These results suggest that our framework balances accuracy and token cost without updating the base model.

6 Conclusion

In this paper, we explored a novel direction of evaluating time series reasoning without post-training base agents, for more efficient and accurate tool calling. Our core contributions are to reformulate time series reasoning as a Markov decision process and to learn an evaluation reward model for reshaping agent actions. We then verify this design through a reinforcement learning framework, TimesEval, built upon episode relative advantage learning and evolving policy optimization for time series reasoning. Extensive experiments on 15 datasets and 4 tasks demonstrate the effectiveness and efficiency of TimesEval. Moreover, efficient tool calling is a crucial topic for agents, but is usually trained through costly post-training, whereas TimesEval achieves it with a lightweight evaluation reward model that leaves the agent unchanged. Thus, we hope this work can facilitate more future research of tool-augmented LLM agents on time series reasoning.

AI USE STATEMENT

(This section is **required** and does not count toward the page limit.)

In this work, we used generative AI tools to polish the writing, including checking grammar, clarity, and phrasing of the sentences. We have not used AI tools for developing research ideas, designing methods, conducting experiments, or analyzing results. We have reviewed and edited all AI-assisted text to ensure its accuracy and consistency with our findings. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International conference on machine learning*, pp. 22–31. Pmlr, 2017.

Frederick Callaway, Sayan Gul, Paul M Krueger, Thomas L Griffiths, and Falk Lieder. Learning to select computations. *arXiv preprint arXiv:1711.06892*, 2017.

Ching Chang, Yidan Shi, Defu Cao, Wei Yang, Jeehyun Hwang, Haixin Wang, Jiacheng Pang, Wei Wang, Yan Liu, Wen-Chih Peng, et al. A survey of reasoning and agentic systems in time series with large language models. *arXiv preprint arXiv:2509.11575*, 2025.

Wenjie Chen, Wenbin Li, Di Yao, Xuying Meng, Chang Gong, and Jingping Bi. Gtool: Graph enhanced tool planning with large language model. *arXiv preprint arXiv:2508.12725*, 2025a.

Yifei Chen, Guanting Dong, and Zhicheng Dou. Toward effective tool-integrated reasoning via self-evolved preference learning. *arXiv preprint arXiv:2509.23285*, 2025b.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*, 2025.

Jiaxuan Gao, Wei Fu, Minyang Xie, Shusheng Xu, Chuyi He, Zhiyu Mei, Banghua Zhu, and Yi Wu. Unlocking long-horizon agentic search with large-scale end-to-end rl. In *The Fourteenth International Conference on Learning Representations*, 2026.

Tong Guan, Zijie Meng, Dianqi Li, Shiyu Wang, Chao-Han Huck Yang, Qingsong Wen, Zuozhu Liu, Sabato Marco Siniscalchi, Ming Jin, and Shirui Pan. Timeomni-1: Incentivizing complex reasoning with time series in large language models. *arXiv preprint arXiv:2509.24803*, 2025.

Shayan Mohajer Hamidi, Linfeng Ye, and Konstantinos N Plataniotis. Dreamphase: Offline imagination and uncertainty-guided planning for large-language-model agents. In *The Fourteenth International Conference on Learning Representations*, 2026.

Jessica B Hamrick, Andrew J Ballard, Razvan Pascanu, Oriol Vinyals, Nicolas Heess, and Peter W Battaglia. Metacontrol for adaptive imagination-based optimization. *arXiv preprint arXiv:1705.02670*, 2017.

Nicholas Hay, Stuart Russell, David Tolpin, and Solomon Eyal Shimony. Selecting computations: Theory and applications. *arXiv preprint arXiv:1408.2048*, 2014.

Joey Hong, Anca Dragan, and Sergey Levine. Planning without search: Refining frontier llms with offline goal-conditioned rl. *Advances in Neural Information Processing Systems*, 38:10032–10056, 2026.

Qihe Huang, Zhengyang Zhou, Yangze Li, Kuo Yang, Binwu Wang, and Yang Wang. Many minds, one goal: Time series forecasting via sub-task specialization and inter-agent cooperation. *Advances in Neural Information Processing Systems*, 38:166475–166506, 2026.

Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. Tree search for llm agent reinforcement learning. *arXiv preprint arXiv:2509.21240*, 2025.

Yushan Jiang, Wenchao Yu, Geon Lee, Dongjin Song, Kijung Shin, Wei Cheng, Yanchi Liu, and Haifeng Chen. Timexl: Explainable multi-modal time series prediction with llm-in-the-loop. *Advances in Neural Information Processing Systems*, 38:135431–135467, 2026.

Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, et al. Time-llm: Time series forecasting by reprogramming large language models. In *International conference on learning representations*, volume 2024, pp. 23857–23880, 2024.

Ming Jin, Yaxuan Kong, Yuxuan Liang, Chaoli Zhang, Siqiao Xue, Xue Wang, James Zhang, Yi Wang, Haifeng Chen, Xiao-Li Li, et al. Large models for time series and spatio-temporal data: A survey and outlook. *ACM Computing Surveys*, 58(14):1–38, 2026.

Masayuki Kawarada, Tatsuya Ishigaki, and Hiroya Takamura. Prompting for numerical sequences: A case study on market comment generation. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 13190–13200, 2024.

Yaxuan Kong, Yiyuan Yang, Yoontae Hwang, Wenjie Du, Stefan Zohren, Zhangyang Wang, Ming Jin, and Qingsong Wen. Time-mqa: Time series multi-task question answering with context enhancement. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 29736–29753, 2025.

Chengpeng Li, Zhengyang Tang, Ziniu Li, Mingfeng Xue, Keqin Bao, Tian Ding, Ruoyu Sun, Benyou Wang, Xiang Wang, Junyang Lin, et al. Teaching language models to reason with tools. *Advances in Neural Information Processing Systems*, 38:66843–66892, 2026a.

Hao Li, Bowen Deng, Chang Xu, Zhiyuan Feng, Viktor Schlegel, Yu-Hao Huang, Yizheng Sun, Jingyuan Sun, Kailai Yang, Yiyao Yu, et al. Mira: Medical time series foundation model for real-world health data. *Advances in neural information processing systems*, 38:98657–98685, 2026b.

Haobo Li, Zhaowei Wang, Jiachen Wang, YueYa Wang, Alexis Kai Hon Lau, and Huamin Qu. Cllmate: A multimodal benchmark for weather and climate events forecasting. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 17547–17573, 2025a.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. Torl: Scaling tool-integrated rl. *arXiv preprint arXiv:2503.23383*, 2025b.

Zihao Li, Xiao Lin, Zhining Liu, Jiaru Zou, Ziwei Wu, Lecheng Zheng, Dongqi Fu, Yada Zhu, Hendrik Hamann, Hanghang Tong, et al. Language in the flow of time: Time-series-paired texts weaved into a unified temporal narrative. In *International Conference on Learning Representations*, volume 2026, pp. 24437–24484, 2026c.

Penghang Liu, Elizabeth Fons, Svitlana Vyetrenko, Daniel Borrajo, Vamsi K Potluru, and Manuela Veloso. Ts-agent: A time series reasoning agent with iterative statistical insight gathering. In *First Workshop on Foundations of Reasoning in Language Models*, 2025a.

Shikang Liu, Xiren Zhou, and Huanhuan Chen. Multiscale temporal dynamic learning for time series classification. *IEEE Transactions on Knowledge and Data Engineering*, 37(6):3543–3555, 2025b.

Zijia Liu, Peixuan Han, Haofei Yu, Haoru Li, and Jiaxuan You. Time-r1: Towards comprehensive temporal reasoning in llms. *arXiv preprint arXiv:2505.13508*, 2025c.

Junkai Lu, Peng Chen, Xingjian Wu, Yang Shu, Chenjuan Guo, Christian S Jensen, and Bin Yang. Patra: Pattern-aware alignment and balanced reasoning for time series question answering. *arXiv preprint arXiv:2602.23161*, 2026.

Yucong Luo, Yitong Zhou, Mingyue Cheng, Jiahao Wang, Daoyu Wang, Tingyue Pan, and Jintao Zhang. Time series forecasting as reasoning: A slow-thinking approach with reinforced llms. *arXiv preprint arXiv:2506.10630*, 2025.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Cheng Qian, Emre Can Acikgoz, Hongru Wang, Xiusi Chen, Avirup Sil, Dilek Hakkani-Tur, Gokhan Tur, and Heng Ji. Smart: Self-aware agent for tool overuse mitigation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 4604–4621, 2025.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tur, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. *Advances in Neural Information Processing Systems*, 38:105523–105553, 2026.

Shuofei Qiao, Honghao Gui, Chengfei Lv, Qianghuai Jia, Huajun Chen, and Ningyu Zhang. Making language models better tool learners with execution feedback. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3550–3568, 2024.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pp. 9695–9717, 2024.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

Richard S Sutton, Andrew G Barto, and Andrew Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

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

Hongru Wang, Cheng Qian, Wanjun Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. Acting less is reasoning more! teaching model to act efficiently. *arXiv preprint arXiv:2504.14870*, 2025.

Siyuan Wang, Peng Chen, Yihang Wang, Wanghui Qiu, Guo Guo, Bin Yang, and Yang Shu. Unlocking the value of text: Event-driven reasoning and multi-level alignment for time series forecasting. In *International Conference on Learning Representations*, volume 2026, pp. 31182–31210, 2026.

Xinlei Wang, Maike Feng, Jing Qiu, Jinjin Gu, and Junhua Zhao. From news to forecast: Integrating event analysis in llm-based time series forecasting with reflection. *Advances in Neural Information Processing Systems*, 37:58118–58153, 2024.

Anna Winnicki and R Srikant. On the convergence of policy iteration-based reinforcement learning with monte carlo policy evaluation. In *International conference on artificial intelligence and statistics*, pp. 9852–9878. PMLR, 2023.

Xingjian Wu, Junkai Lu, Zhengyu Li, Xiangfei Qiu, Jilin Hu, Chenjuan Guo, Christian S Jensen, and Bin Yang. Timeart: Towards agentic time series reasoning via tool-augmentation. *arXiv preprint arXiv:2601.13653*, 2026.

Shuo Yang, Soyeon Caren Han, Yihao Ding, Shuhe Wang, and Eduard Hoy. Tooltree: Efficient llm agent tool planning via dual-feedback monte carlo tree search and bidirectional pruning. *arXiv preprint arXiv:2603.12740*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Kenny Young and Richard S Sutton. Understanding the pathologies of approximate policy evaluation when combined with greedification in reinforcement learning. *arXiv preprint arXiv:2010.15268*, 2020.

Fangxu Yu, Hongyu Zhao, and Tianyi Zhou. Ts-reasoner: Aligning time series foundation models with llm reasoning. *arXiv preprint arXiv:2510.03519*, 2025.

Fangxu Yu, Xingang Guo, Lingzhi Yuan, Haoqiang Kang, Hongyu Zhao, Lianhui Qin, Furong Huang, Bin Hu, and Tianyi Zhou. Tsrbench: A comprehensive multi-task multi-modal time series reasoning benchmark for generalist models. *arXiv preprint arXiv:2601.18744*, 2026.

Yangyang Yu, Zhiyuan Yao, Haohang Li, Zhiyang Deng, Yuechen Jiang, Yupeng Cao, Zhi Chen, Jordan W Suchow, Zhenyu Cui, Rong Liu, et al. Fincon: A synthesized llm multi-agent system with conceptual verbal reinforcement for enhanced financial decision making. *Advances in Neural Information Processing Systems*, 37:137010–137045, 2024.

Dharunish Yugeswardeenoo, Kevin Zhu, and Sean O'Brien. Question-analysis prompting improves llm performance in reasoning tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pp. 402–413, 2024.

Regina Zhang, Si Qi Goh, Xingsheng Chen, Zongru Li, Honggang Wen, Jiale Guo, Menglin Yang, Hongzhi Yin, Qiang Yang, SM Yiu, et al. From prompts to agents: A comprehensive survey of llm-driven time series analysis. *Available at SSRN 6614598*, 2025.

Siru Zhong, Weilin Ruan, Ming Jin, Huan Li, Qingsong Wen, and Yuxuan Liang. Time-vlm: Exploring multimodal vision-language models for augmented time series forecasting. *arXiv preprint arXiv:2502.04395*, 2025.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

Jiahui Zhou, Dan Li, Lin Li, Zhuomin Chen, Shunyu Wu, Haozheng Ye, Jian Lou, and Costas J Spanos. Enhancing llm reasoning for time series classification by tailored thinking and fused decision. *arXiv preprint arXiv:2506.00807*, 2025.

Jiahui Zhou, Dan Li, Boxin Li, Xiao Zhang, Erli Meng, Lin Li, Zhuomin Chen, Jian Lou, and See-Kiong Ng. Time series reasoning via process-verifiable thinking data synthesis and scheduling for tailored llm reasoning. *arXiv preprint arXiv:2602.07830*, 2026.

Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan Rossi, Somdeb Sarkhel, and Chao Zhang. Toolchain*: Efficient action space navigation in large language models with a* search. In *International Conference on Learning Representations*, volume 2024, pp. 4524–4549, 2024.

A Extended Experiments

A.1 Efficiency Analysis of Tool Execution

As shown in Table 3, TimesEval achieves the highest accuracy in every task category across all four backbones, with relative improvements of up to 29.2% over the strongest baseline. The gains are most pronounced on Reasoning, which suggests that reasoning questions benefit most from evaluating each tool call before execution. Meanwhile, TimesEval uses fewer tool calls than ReAct and reduces tool calls by 60.4% on average compared with ToolTree, indicating that its accuracy gains do not come from more extensive tool use. Notably, fewer tool calls alone do not improve accuracy, as LATS invokes no tools on TS-Reasoner-7B yet yields the lowest average accuracy.

A.2 Full Results against Post-Trained Models

Table 4 compares TimesEval with RL-trained time series models, where TimesEval builds on Qwen3-8B without updating its parameters. TimesEval achieves the highest overall accuracy, with a relative improvement of 55.0% over VeriTime, the strongest RL-trained model, and ranks first on 14 of the 15 tasks. The gains are large on reasoning, prediction, and decision-making tasks such as DR, EP, and QuantDM, where the RL-trained models remain far behind. Meanwhile, TimesEval reduces the average token cost by about half compared with VeriTime and Time-R1, both of which consume more than twice as many tokens as TimesEval on tasks such as CD, AR, DR, and TSF. Although PATRA uses slightly fewer tokens overall, its accuracy is substantially lower, suggesting that evaluating tool calls with a lightweight evaluation reward model achieves a better balance between accuracy and cost than post-training the base model.

A.3 Ablation Study Design

The following variants examine the contributions of evaluator inputs, training supervision, policy evolution, and constrained selection. The comparison protocol uses the same frozen base agent, tool set, data splits, and deployment budgets. Candidate policies are selected on the validation set and evaluated on the same held-out questions. Learning-based comparisons use matched training seeds to account for differences in rollout requirements.

- **Always-Commit.** The reference policy executes every proposed tool call while retaining the base agent's answer-generation and termination rules under the common interaction budget. This comparison measures the overall effect of introducing the evaluator.
- **Unpaired Training.** For each training state, a feasible action is sampled and evaluated through a complete rollout under the same continuation policy used for paired supervision. A model with comparable capacity is trained to predict $Q(s, a)$ from the resulting returns, using the same frozen state encoder. At deployment, actions are compared through $Q(s, a) - Q(s, \mathsf{Commit})$. Under a matched rollout-generation budget, this variant can cover more distinct states, with sampling balanced across feasible actions. It tests whether comparing actions from a common state improves learning beyond single-branch returns.
- **Evidence Masked.** Historical tool-output content is removed from the evaluator input, while the question, time series, pending tool call, executed-call records, and budget information are retained. The evaluator is retrained with this representation, and the base agent continues to receive the complete evidence history. This ablation tests the contribution of directly observed evidence beyond structural cues such as call counts; pending proposals may still convey information derived from previous observations.
- **Fixed Behavior.** Newly visited states are collected by the fixed Always-Commit policy in every round. Relative advantage targets are still recomputed under the current continuation policy μ_{m-1}, followed by evaluator training and constrained policy selection. This variant tests the contribution of collecting states induced by the evolving evaluation policy.
- **w/o Policy Evolution.** Training states and paired targets are generated under Always-Commit. The evaluator is trained in a single offline stage, selected on the validation set, and then frozen. A comparable total rollout-generation budget is allocated to this variant so that the comparison tests iterative state collection and target estimation rather than additional training data.

Table 3: Task-wise accuracy and efficiency comparison. Acc. denotes category-level accuracy (%) derived from Table 1, and TC denotes the average number of tool calls per sample.

Method	Perception		Reasoning		Prediction		Decision	
	Acc. ↑	TC ↓	Acc. ↑	TC ↓	Acc. ↑	TC ↓	Acc. ↑	TC ↓
Qwen3-8B								
ReAct	57.8	0.30	45.0	1.52	42.3	0.45	50.8	0.49
ToolTree	56.4	1.33	48.1	2.08	50.1	2.68	49.6	1.43
LATS	58.2	1.06	43.7	1.33	43.1	0.48	49.8	0.66
Ours	60.3	0.29	51.8	1.29	51.8	0.41	51.5	0.34
TimeOmni-1-7B								
ReAct	55.0	0.14	34.7	0.83	35.2	0.23	48.6	0.47
ToolTree	55.2	0.40	37.2	1.47	32.5	1.58	49.6	1.40
LATS	55.8	1.60	40.9	1.54	38.9	1.14	45.7	1.01
Ours	58.4	0.12	46.7	0.78	44.7	0.23	51.7	0.42
Time-MQA-7B								
ReAct	50.7	0.47	44.1	2.54	33.9	1.98	45.3	2.79
ToolTree	50.0	1.66	44.3	2.68	34.7	2.38	46.3	2.09
LATS	50.2	1.39	29.8	1.36	28.8	0.53	42.1	1.23
Ours	55.1	0.38	48.6	2.49	36.9	0.42	48.6	1.28
TS-Reasoner-7B								
ReAct	48.0	0.10	30.0	0.35	36.7	0.08	42.3	0.54
ToolTree	50.0	0.26	30.5	0.45	35.4	0.10	43.9	1.08
LATS	45.4	0.00	17.4	0.00	36.2	0.00	33.7	0.00
Ours	60.4	0.10	39.4	0.26	39.3	0.05	49.9	0.45

Table 4: Accuracy (%) and average tokens per sample (k) of RL-trained models on TSRBench.

Model	Perception				Reasoning							Prediction		Decision		Overall
	PR	NU	AD	CA	ER	CD	AR	TR	NR	DR	IR	TSF	EP	QualDM	QuantDM	
PATRA	36.8	41.4	43.4	44.2	26.8	28.0	69.3	23.8	36.0	26.0	28.0	32.8	14.8	36.4	16.8	33.6
Tokens (k)	0.2	0.2	0.2	0.2	0.3	0.9	6.7	4.0	0.3	4.8	0.3	4.1	0.3	0.7	0.5	1.6
Time-R1	44.0	49.4	39.5	36.3	26.0	19.6	60.0	21.2	24.0	16.8	35.0	9.6	15.6	23.6	16.0	29.1
Tokens (k)	1.6	1.5	1.9	2.8	2.4	6.2	6.9	5.3	2.5	5.6	2.1	8.5	1.4	4.6	2.2	3.7
VeriTime	57.6	54.0	47.3	60.2	12.0	30.4	69.3	21.2	16.0	25.6	28.0	30.8	31.6	30.4	29.2	36.2
Tokens (k)	1.3	1.4	1.9	2.3	2.5	6.4	6.8	5.2	1.5	5.4	2.0	7.3	1.2	4.4	2.1	3.5
TimesEval	58.4	70.1	49.2	70.7	24.0	68.3	76.7	34.6	40.0	90.0	38.0	47.7	69.0	37.3	67.0	56.1
Tokens (k)	1.7	1.5	1.5	1.7	1.5	2.7	2.2	4.5	0.5	1.5	1.8	2.0	1.4	1.8	1.4	1.8

- **w/o Accuracy Constraint.** The final candidate models and their training procedures are retained, but the accuracy requirement is removed from the final validation selection step. The candidate with the lowest mean weighted cost is selected. This comparison isolates the role of the accuracy constraint in final policy selection; earlier evolution rounds follow the full method's protocol.
- **w/o Cost-aware Return.** Setting $\lambda_c = 0$ makes each branch return equal to final-answer correctness. Relative advantages are still computed against Commit, and evaluator training and policy evolution use these modified targets. The global accuracy constraint and cost-based validation selection are retained. This ablation tests whether including interaction cost in local training targets improves the resulting policy.

A.4 Analysis of Tool Calls and Evaluation Decisions

Intervention Quality across Rounds. Figure 7 compares the interventions made in the initial round ($k = 0$), where no tool output has been observed, and in the second round ($k = 1$). As shown in the left panel, the share of necessary interventions increases from 59% to 78%, while the share of harmful interventions decreases from 39% to 15%. The right panel shows a consistent trend across margins: in the second round, the necessary rate stays around 70% to 80% as the intervention coverage grows beyond 80%, whereas in the initial round it drops below 50% once the coverage exceeds about 40%. These results suggest that the evaluator judges tool calls more reliably as evidence accumulates.

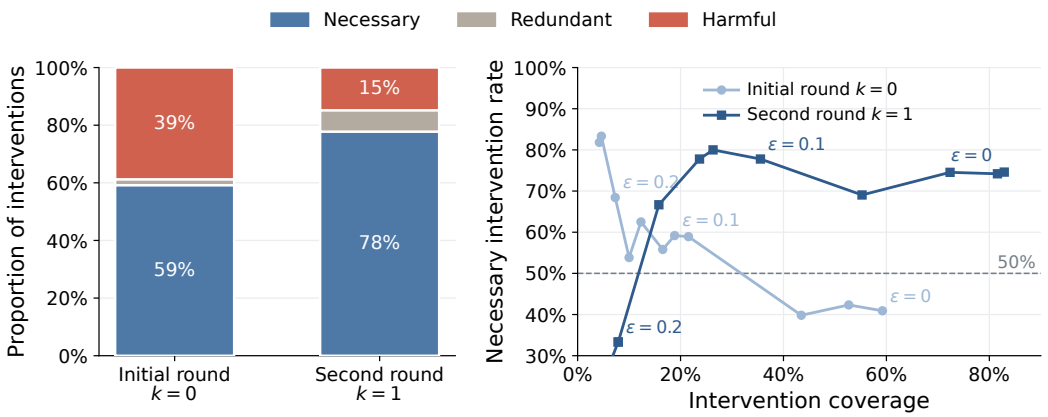


Figure 7: Interventions in the initial and second rounds. **Left**: proportion of necessary, redundant, and harmful interventions. **Right**: necessary rate versus intervention coverage under different margins ϵ. Interventions are more often necessary and less often harmful in the second round.

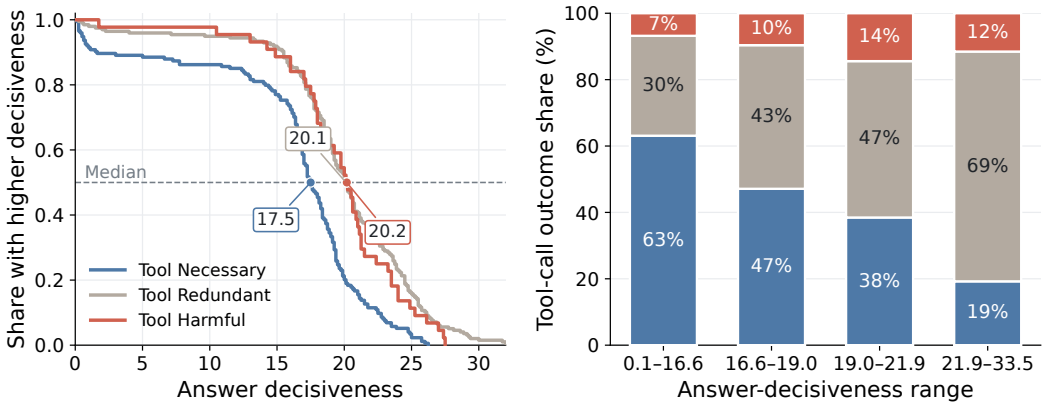


Figure 8: Answer decisiveness and tool-call outcomes. **Left**: distribution of answer decisiveness for necessary, redundant, and harmful tool calls, with medians marked. **Right**: outcome share of tool calls in each decisiveness range. Tool calls are more often redundant when the agent is already decisive about its answer.

Tool Call and Answer Decisiveness. To examine when a tool call is useful, we compare two answers of the base agent at each evaluation state: one produced directly from the current evidence, and one produced after executing the proposed tool call. A tool call is necessary if it turns a wrong answer into a correct one, harmful if it turns a correct answer into a wrong one, and redundant if the correctness is unchanged. We further measure the *answer decisiveness* of the direct answer as the log-probability margin between the top two answer options, where larger values indicate that the agent more strongly favors a single option given the current evidence.

As shown in Figure 8 (left), necessary tool calls occur at lower decisiveness, with a median of 17.5, than redundant and harmful ones, whose medians are 20.1 and 20.2. The right panel divides the tool calls into four decisiveness ranges: as decisiveness increases, the share of necessary calls decreases from 63% to 19%, while the share of redundant calls increases from 30% to 69%. These results indicate that a tool call is more likely to add useful evidence when the agent is uncertain about its answer and tends to be redundant once the agent is already decisive, which supports evaluating each tool call based on the current state.

B BENCHMARK DETAILS

TSRBench (Yu et al., 2026) contains 4,125 multiple-choice problems collected from 14 domains, which are organized into four dimensions and 15 tasks. Perception evaluates the recognition of fundamental patterns and properties of time series, Reasoning evaluates the derivation of conclusions from temporal patterns and prior knowledge, Prediction evaluates the anticipation of future values and events, and Decision-Making evaluates decisions made from both time series and context. The problems are built from real-world data in public repositories and synthetic data generated by simulations, and their ground-truth answers are derived by code or rule-based extraction and then verified. Time series forecasting is also posed as multiple-choice questions, a format shown to yield trends consistent with open-ended numerical forecasting, and 58.1% of the problems involve multivariate time series. Table 5 lists the abbreviation, full name, and proportion of each task.

Table 5: Task taxonomy of TSRBench. Proportion denotes the percentage of problems in each task.

Dimension	Abbr.	Task	Proportion (%)
Perception	PR	Pattern Recognition	9.0
	NU	Noise Understanding	2.1
	AD	Anomaly Detection	3.1
	CA	Comparative Analysis	2.7
Reasoning	ER	Etiological Reasoning	8.5
	CD	Causal Discovery	7.5
	AR	Abductive Reasoning	3.6
	TR	Temporal Relation Reasoning	3.9
	NR	Numerical Reasoning	9.7
	DR	Deductive Reasoning	6.1
	IR	Inductive Reasoning	2.4
Prediction	TSF	Time Series Forecasting	17.5
	EP	Event Prediction	8.7
Decision-Making	QualDM	Qualitative Decision-Making	8.1
	QuantDM	Quantitative Decision-Making	7.3

C ADDITIONAL METHOD DETAILS

We introduce the definitions and parameters of TimesEval omitted from the main text, including the interaction cost, the feasible evaluation decisions, the return, and the evaluation reward model.

Interaction Cost. We define the interaction cost of an execution trace ξ as:

$$C(\xi) = N_{\text{tool}}(\xi) + \lambda_{\text{LLM}}\, N_{\text{LLM}}(\xi), \tag{7}$$

where $N_{\text{tool}}(\xi)$ is the number of executed tool calls, $N_{\text{LLM}}(\xi)$ is the number of LLM calls made by the base agent, and $\lambda_{\text{LLM}} = 1$ weights LLM calls relative to tool calls. A proposal rejected by REPLAN is never executed and is excluded from N_{tool}, whereas both the LLM call that produces it and the LLM call that produces the new proposal are included in N_{LLM}. Since each trace contains at most $T_{\max} = 2$ executed tool calls, at most $N_{\text{rep}} = 1$ replan before each execution, and one answering call, its cost satisfies $C(\xi) \le T_{\max} + \lambda_{\text{LLM}}\big(T_{\max}(N_{\text{rep}} + 1) + 1\big) = 7$. We set $C_{\max} = 9$, which upper-bounds the cost and keeps the normalized cost $C(\xi)/C_{\max}$ within $[0, 1]$. With the cost weight $\lambda_c = 0.05$, the cost penalty $\lambda_c C(\xi)/C_{\max}$ never exceeds 0.04, so the correctness term dominates the return and the cost term mainly distinguishes decisions that lead to the same correctness.

Feasible Decisions. The base agent executes at most $T_{\max} = 2$ tool calls per problem, and each problem involves at most five evaluation rounds. COMMIT and STOP are always feasible, whereas REPLAN becomes infeasible when either of the following holds: the consecutive replanning count has reached $N_{\text{rep}} = 1$, or adding the proposed tool to the exclusion set would exclude all tools in its category. Once the $T_{\max}$-th tool call is executed, evaluation is skipped and the base agent is directly asked to answer. Upon REPLAN, the exclusion set $B^{(k)}$ is provided to the base agent in two

ways: the prompt lists the rejected tools and asks the agent to choose another tool or answer directly, and the excluded tools are removed from the enumerated options of the `next_action` field in the structured output, which retains only the remaining tools and `answer_now`. The exclusion set is cleared after each executed tool call. If the new proposal after REPLAN cannot be parsed, the decision falls back to COMMIT and the original proposal is executed.

Return. For an episode that starts from an evaluation state s with decision a and then follows the continuation policy μ_{m-1}, the return is:

$$G(a) = \mathbb{I}[\hat{y} = y] - \lambda_c \frac{C_{\text{suffix}}}{C_{\max}}, \tag{8}$$

where the correctness term indicates whether the final answer $\hat{y}$ of the trajectory equals the ground truth y, and C_{suffix} is the interaction cost incurred from s onward, namely the number of tool calls plus λ_{LLM} times the number of LLM calls, excluding the cost before s. Since all branches from the same state share the same prefix, excluding the prefix cost leaves the relative advantage $G(a) - G(\mathsf{Commit})$ unchanged. The values of λ_c, $C_{\max}$, and λ_{LLM} follow the interaction cost above.

Evaluation Reward Model. The encoder E is the same model as the base agent. Given an evaluation state serialized into at most 1,536 tokens, we take the last-layer hidden state of the last non-padding token, normalize it, reduce it to 256 dimensions with a fixed projection, and concatenate it with compact structural features of the state. The evaluation reward model A_ω is an MLP with hidden sizes 64 and 32, which outputs the predicted advantage $A_\omega(E(s), a)$ of each feasible decision relative to COMMIT, with $A_\omega(E(s), \mathsf{Commit}) = 0$. It is trained for 120 epochs with three random seeds. At inference, we select $a^* = \arg\max_{a \in \mathcal{A}(s) \setminus \{\mathsf{Commit}\}} A_\omega(E(s), a)$ and execute a^* if $A_\omega(E(s), a^*) \geq \epsilon$, and COMMIT otherwise. The margin ϵ is selected on the validation set from $\{0, 0.05, 0.1, 0.2\}$, which yields $\epsilon = 0$, 0, and 0.1 for the three seeds.

D TRAINING CONFIGURATION AND HYPERPARAMETERS

Data Collection. The data splits is summarized in Table 6. The training set contains 900 problems with 60 problems per task, which are evenly divided into three evolution rounds so that each round introduces 300 new problems. The validation set contains 300 problems with 20 problems per task, which are divided into two disjoint halves: $\mathcal{D}_{\text{val}}$ for policy selection and the other half for analyzing evaluation decisions. A test set of about 1,078 problems is held out for final evaluation.

In each evolution round, we run the current evaluation policy on the 300 new problems and collect every evaluation state in which a tool call has been proposed but not yet executed. From each state, we roll out every feasible decision among COMMIT, REPLAN, and STOP from the same prefix and continue each branch with the current policy until termination. We compute the return of each branch as defined above and subtract the return of COMMIT to obtain the advantage target of each decision. The state pool is accumulated across rounds, and the targets of earlier states are recomputed with continuations under the current policy. For TimesEval, the pool contains about 320, 650, and 1,010 states after the three rounds. In total, the 1,200 training and validation problems yield about 9,540 unique LLM requests, 6,180 trajectories, and 3,120 reachable evaluation states, with every feasible decision rolled out at each state.

Table 6: Data splits of TSRBench used in our experiments.

Split	#Problems	Per task	Usage
Training	900	60	Evaluation reward model training (300 per round)
Validation	300	20	Policy selection ($\mathcal{D}_{\text{val}}$, 150) and analysis (150)
Test	$\sim$1,078	–	Final evaluation

Hyperparameters. We summarize the key hyperparameters of TimesEval in Table 7, covering the interaction cost, the evaluation decisions, the evaluation reward model, and the evolving policy optimization. Values introduced in Appendix C are repeated here for completeness.

Table 7: Hyperparameters.

Group	Hyperparameter	Value
Interaction budget	Max executed tool calls $T_{\max}$	2
	Max consecutive replans N_{rep}	1
	Max evaluation rounds	5
Cost and return	LLM-call weight λ_{LLM}	1.0
	Cost weight λ_c	0.05
	Cost normalizer $C_{\max}$	9
Policy selection	Margin candidates ϵ	$\{0, 0.05, 0.1, 0.2\}$
	Accuracy tolerance δ	2 pp
	Evolution rounds	3
	New questions per round	300
State encoder	Encoder	Same as base agent
	Projection dimension	256
	Max input length	1536 tokens
Evaluation reward model	MLP hidden sizes	$[64, 32]$
	Epochs	120
	Optimizer	AdamW
	Learning rate	1×10^{-3}
	Weight decay	1×10^{-4}
	Batch size	64
	Training seeds	3
Decoding	Temperature	0.0
	Top-p	0.95
	Thinking mode	Disabled
	Max tokens (proposal / answer)	256 / 512
	Decoding seed	42

E TOOL SPECIFICATIONS

We extend the time series analysis tools from TSRBench (Yu et al., 2026), where most tools take the raw time series as input and returns a structured statistical summary as its observation. Before analysis, non-numeric artifacts are removed and each series is converted to a floating-point array. Table 8 summarizes the tools, which cover descriptive statistics, trend analysis, peak and valley detection, change point detection, and multivariate comparison.

Table 8: Time series analysis tools used in our experiments. The first five components follow TSR-Bench (Yu et al., 2026), and the remaining tools are added for different tasks.

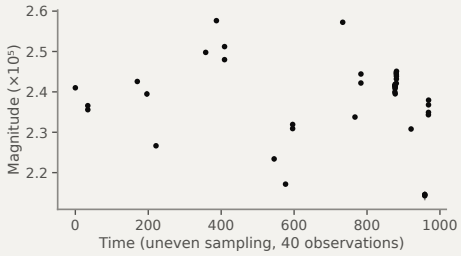
Component	Method	Output
Descriptive statistics	Statistical moments via `scipy.stats`	Mean, std, range, median, variance, skewness, and kurtosis
Trend analysis	OLS regression $x_t = \beta_1 t + \beta_0 + \epsilon$	Slope β_1, R^2, and trend strength by Pearson $\lvert r\rvert$ (strong > 0.7, moderate > 0.4, else weak)
Peak and valley detection	`scipy.signal.find_peaks` with prominence $P = 0.5\,\sigma_X$	Indices of local maxima and minima
Change point detection	First-order difference $\Delta x_t = x_t - x_{t-1}$	Steps with $\lvert\Delta x_t\rvert > 2\,\mathrm{std}(\Delta X)$
Multivariate comparison	Pearson correlation, normalized cross-correlation, and Welch's t-test	$\rho_{X,Y}$ with p-value, lag $\tau_{\text{best}} = \arg\max_\tau (X \star Y)(\tau)$, and mean-difference significance ($p < 0.05$)
Noise level estimate	Std of one series, or std of first-difference residuals of two series	Noise level, or the noisier series
Pairwise series comparison	Variance, amplitude, and least-squares slope of two co-measured series	Larger statistic, or whether trend directions agree
Multi-axis motion statistics	Per-channel mean, std, and range	Intensity and variability of each channel
Cyclical projection	Dominant period from the FFT periodogram (`scipy.signal.periodogram`) and phase-matched extrapolation	Dominant period and projected value at the queried time
Lomb-Scargle periodogram	`astropy` Lomb-Scargle on the first (time) and second (value) columns; period $= 1/f^*$ at peak power, amplitude $= (x_{\max} - x_{\min})/2$	Estimated period and amplitude
Catch22 features	Six catch22 features of the first column via `pycatch22`: histogram mode, longest stretch above mean, first $1/e$ ACF crossing, low-frequency power, mean, and std	Feature values, plus knowledge requirements from problem metadata if available
Point-value consistency check	Score change at the critical moment vs. points implied by each option (free throw 1, two-point 2, three-point 3, no made shot 0); time series not used	Consistency of each option with the observed change
Lorenz integration	Forward Euler integration of the Lorenz system for 120 steps, with σ, ρ, β, Δt, and initial state parsed from the question	Trajectory $x(t)$ and RMSE of each option, in ascending order
Weather trend statistics	Latest value, recent mean, and trend direction of each weather variable	Per-variable trends, without a computed verdict

F CASE STUDY

Figures 9–12 present four cases that cover all three evaluation decisions. In Case 1, the periodogram already indicates a long-period variable, and TimesEval stops the subsequent feature extraction, which is applied to the time column and adds no evidence, reaching the same correct answer with one fewer tool call. In Case 2, the proposed tool returns summary statistics that do not help estimate the rate constant, after which Always-Commit falls back on a two-point estimate and answers wrongly, whereas TimesEval stops before this call and answers correctly at half the cost. In Case 3, the consistency check cannot distinguish two options and misleads Always-Commit, while TimesEval rejects it through REPLAN and the agent answers correctly from the rise in win probability, at the cost of one extra LLM call. In contrast, Case 4 requires numerically integrating the Lorenz system, where the Fixed-Behavior ablation stops the necessary tool call and answers wrongly, but TimesEval commits it and identifies the correct trajectory, showing that TimesEval rejects low-value tool calls rather than suppressing tool use.

Case 1: Stop saves cost, answer unchanged

INPUT



Question

The following table contains light curve data from a star. Based on an analysis of the light curve's period, amplitude, and shape, Which type of variable star generates it with the largest probability?

Choices

A. Rotational variables
B. RR Lyrae
C. Cepheids
D. Long-period variables
E. Delta Scuti

ALWAYS-COMMIT

Round 1

Agent proposes `periodogram_unevenly_sampled` Commit

"The timestamps are unevenly sampled, so I should use the `periodogram_unevenly_sampled` tool to estimate the period."

```
Lomb-Scargle periodogram: estimated period = 421.2
(same time units as input), amplitude = 2.168e+04 mag.
```

Round 2

Agent proposes `generic_series_features` Commit

"To further distinguish between the variable star types, I need to analyze the shape of the light curve and its characteristics."

```
Computed statistical features for 'Time': {...,
'DN_Mean': 689.1795, 'DN_Spread_Std': 303.1425}.
→ computed on the Time column, not on the magnitudes
```

✓ **Answer D (correct)** cost 5 = 2 tool + 3 LLM calls

"...long-period variables (D) typically have periods ranging from days to months, which aligns with the estimated period."

TIMESEVAL (OURS)

Round 1

Agent proposes `periodogram_unevenly_sampled` Commit

"The timestamps are unevenly sampled, so I should use the `periodogram_unevenly_sampled` tool to estimate the period."

```
Lomb-Scargle periodogram: estimated period = 421.2
(same time units as input), amplitude = 2.168e+04 mag.
```

Round 2

Agent proposes `generic_series_features` Stop

"To further distinguish between the variable star types, I need to analyze the shape of the light curve and its characteristics."

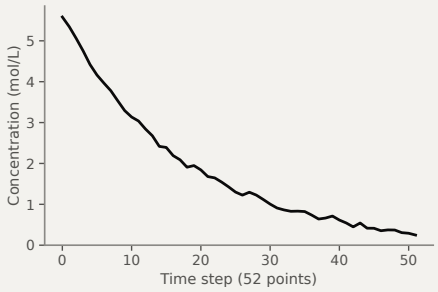
✓ **Answer D (correct)** cost 4 = 1 tool + 3 LLM calls

"This period is relatively long, and the amplitude is large."

Figure 9: Case study of STOP. After the periodogram already indicates a long-period variable, Always-Commit executes an extra tool that adds no useful evidence, while TimesEval stops and gives the same correct answer with one fewer tool call.

Case 2: Stop avoids interference, wrong to right

INPUT



Question

The following time series shows the concentration (in mol/L) of a reactant over time during a chemical reaction. Based on the concentration decay pattern, determine the reaction order (zero, first, or second) and calculate the rate constant k. For zero-order: $C = C_0 - kt$; First-order: $C = C_0e^{(-kt)}$; Second-order: $1/C = 1/C_0 + kt$. Assume this is a first-order reaction and calculate k.

Choices

A. 0.0536
B. 1.752
C. 0.0204
D. 0.0292

ALWAYS-COMMIT

Round 1

Agent proposes `generic_series_features` Commit

"To determine the reaction order and calculate the rate constant k for a first-order reaction, I need to analyze the concentration data using the first-order equation: $C = C_0e^{(-kt)}$. This involves plotting ln(C) vs. time and finding the slope, which will be -k. The generic_series_features tool can help compute summary statistics of the concentration series, including the natural logarithm of the concentrations, which is necessary for this analysis."

```
Computed statistical features for 'Reactant
Concentration (mol/L)': {..., 'DN_Mean': 1.8419,
'DN_Spread_Std': 1.4941}. This problem likely
requires: ['Reaction kinetics', 'Rate laws',
'Integrated rate equations'].
→ summary statistics only; no ln(C) values or fitted slope
```

✗ **Answer A (wrong)** cost 4 = 1 tool + 3 LLM calls

"...since the question assumes the reaction is first-order, we can directly calculate ... the initial concentration (first value in the series, 5.583 mol/L) ... Using the final concentration (0.248 mol/L) and the corresponding time (49 time units), we calculate ... The closest choice is A (0.0536)."

TIMESEVAL (OURS)

Round 1

Agent proposes `generic_series_features` Stop

"To determine the reaction order and calculate the rate constant k for a first-order reaction, I need to analyze the concentration data using the first-order equation: $C = C_0e^{(-kt)}$. This involves plotting ln(C) vs. time and finding the slope, which will be -k. The generic_series_features tool can help compute summary statistics of the concentration series, including the natural logarithm of the concentrations, which is necessary for this analysis."

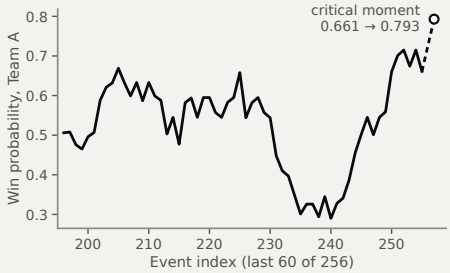
✓ **Answer D (correct)** cost 2 = 0 tool + 2 LLM calls

"...so we use the first-order equation: $C = C_0e^{(-kt)}$. To find the rate constant k, we can take the natural logarithm of both sides: $\ln(C) = \ln(C_0) - kt$. This means plotting ln(C) vs. time should yield a straight line with slope -k. Since the data is provided, we can calculate ln(C) for each concentration value and perform a linear regression to find the slope. The slope will be approximately -k. From the given choices, the correct value of k is 0.0292."

Figure 10: Case study of STOP that corrects the answer. The proposed tool returns only summary statistics that do not help estimate k, and Always-Commit then answers wrongly, while TimesEval stops before this call and answers correctly at half the cost.

Case 3: Replan, wrong to right

INPUT



Critical moment

score went from 86 - 81 at 9:13-4th to 86 - 81 at 8:48-4th (win probability before: {'Team A': 0.661, 'Team B': 0.339}, after: {'Team A': 0.793, 'Team B': 0.207}).

Question

Based on the game's context and the data provided up to this point, what was the most likely event to occur?

Game events (last 3 of 256)

9:17-4th: Player 11 (Team A) misses 23-foot three point shot
9:14-4th: Player 3 (Team A) offensive rebound
9:13-4th: Player 3 (Team A) makes 2-foot dunk | Team B Full timeout | Player 4 (Team B) enters the game for Player 7 (Team B)

Choices

A. Player 8 (Team A) enters the game for Player 18 (Team A)
B. Player 10 (Team B) makes 22-foot three point jumper (Player 2 (Team A) assists)
C. Player 7 (Team A) blocks Player 3 (Team B)'s two point shot
D. Player 2 (Team A) makes 1-foot dunk (Player 12 (Team A) assists)

ALWAYS-COMMIT

Round 1

Agent proposes `point_value_consistency_check` Commit

"...The `point_value_consistency_check` tool is appropriate here as it can compare the implied numeric values of the event descriptions against the observed changes in win probability."

```
Observed score change at the critical moment: 86 - 81
-> 86 - 81 (implied points scored = 0).
Option A: implies 0 point(s) -> consistent
Option B: implies 3 point(s) -> INCONSISTENT
Option C: implies 0 point(s) -> consistent
Option D: implies 2 point(s) -> INCONSISTENT
→ A and C are both consistent
```

✗ **Answer A (wrong)** cost 4 = 1 tool + 3 LLM calls

"...Since the score did not change, the most likely event is a substitution, making Option A the most plausible choice."

TIMESEVAL (OURS)

Round 1

Agent proposes `point_value_consistency_check` Replan

"...The `point_value_consistency_check` tool is appropriate here as it can compare the implied numeric values of the event descriptions against the observed changes in win probability."

→ not executed; excluded from the next proposal

Round 2

Agent proposes `periodogram_unevenly_sampled` Commit

"...However, it was previously rejected, possibly due to lack of observed numeric values."

```
Lomb-Scargle periodogram: estimated period = 0.1429
(same time units as input), amplitude = 40.5 mag.
```

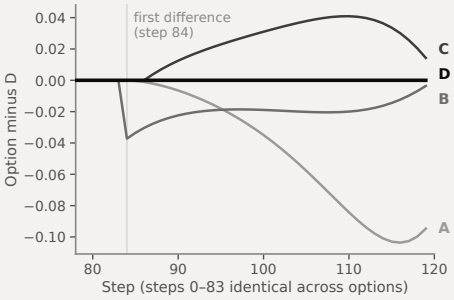
✓ **Answer C (correct)** cost 5 = 1 tool + 4 LLM calls

"...with Team A's probability increasing from 0.661 to 0.793. ... only option C involves a defensive stop (a block), which aligns with the observed increase in Team A's win probability."

Figure 11: Case study of REPLAN that corrects the answer.

Case 4: Commit keeps a necessary tool

INPUT



Choices (120 values each)

A. [1.68, 1.6711, 1.7063, ..., -13.1737]
B. [1.68, 1.6711, 1.7063, ..., -13.0826]
C. [1.68, 1.6711, 1.7063, ..., -13.0647]
D. [1.68, 1.6711, 1.7063, ..., -13.079]

Question

A time series x(t) is generated by a three-dimensional chaotic physical system (the Lorenz Attractor). The system's state [x(t), y(t), z(t)] evolves according to the following deterministic set of coupled differential equations, approximated by the discrete-time Euler method. Extreme precision is required as even minor deviations can lead to incorrect results.
System Rules:
$x(t) = x(t-\Delta t) + \Delta t * \sigma * (y(t-\Delta t) - x(t-\Delta t))$
$y(t) = y(t-\Delta t) + \Delta t * (x(t-\Delta t) * (\rho - z(t-\Delta t)) - y(t-\Delta t))$
$z(t) = z(t-\Delta t) + \Delta t * (x(t-\Delta t) * y(t-\Delta t) - \beta * z(t-\Delta t))$
For this specific instance, the parameters are:
σ (Sigma) = 9.88
ρ (Rho) = 28.12
β (Beta) = 2.654
Δt (time step) = 0.01
The initial conditions at t=0 are:
$(x_0, y_0, z_0) = (1.68, 1.59, 1.15)$
Which of the following time series for x(t) perfectly and completely represents the evolution of this system for all 120 steps?

FIXED-BEHAVIOR

Round 1
Agent proposes `lorenz_ode_fit` Stop

"...The lorenz_ode_fit tool is designed for this exact scenario: it numerically integrates the system forward and compares each candidate's numeric trajectory to the result."

✗ **Answer C (wrong)** cost 2 = 0 tool + 2 LLM calls

"...By comparing the initial values and the first few steps of the time series, we can identify the correct sequence. The first few values in all choices match the initial condition (1.68) and the first step (1.6711)."

TIMESEVAL (OURS)

Round 1
Agent proposes `lorenz_ode_fit` Commit

"...The lorenz_ode_fit tool is designed for this exact scenario: it numerically integrates the system forward and compares each candidate's numeric trajectory to the result."

```
Computed x(t) trajectory via forward-Euler integration
of the given equations (σ=9.88, ρ=28.12, β=2.654,
Δt=0.01, x0,y0,z0=(1.68,1.59,1.15)):
x(t)[:10] = [1.68, 1.67111, 1.70629, ...]
RMSE of each candidate option against this computed
trajectory (lowest = most consistent):
Option D: RMSE = 2.98577e-06
Option B: RMSE = 0.0113549
Option C: RMSE = 0.0157308
Option A: RMSE = 0.0331021
```

✓ **Answer D (correct)** cost 4 = 1 tool + 3 LLM calls

"...The results show that Option D has the lowest RMSE (2.98577e-06), which indicates it is the most accurate and consistent with the true trajectory of the system."

Figure 12: Case study of COMMIT.