RITS: REASONING WITH IDENTIFIABLE TRACES FOR TIME-SERIES ANALYSIS AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

In domains such as finance, healthcare, and industrial monitoring, many decisions rely on time-series analysis. Tool-using agents promise to automate such analyses posed as natural-language questions that require a multi-step workflow of analysis tools. However, training and evaluating agents that execute such analyses faithfully is challenging because natural analysis questions often underspecify the intended procedure. Specifically, the same question can be addressed with different tools and parameter bindings, and these choices can yield different answers. There is then no single correct answer to train on or to grade against, and no unique reference workflow exists to check a produced analysis against. We present RITS, a framework that constructs trace-identifiable multi-hop time-series questions and trains and evaluates agents against their reference workflows. We build every question to admit a unique executable workflow within a modeled tool space, and refer to that workflow as a trace. A produced analysis can thus be scored against a single reference. To this end, RITS instantiates templates structured as directed acyclic graphs (DAGs) over 308 parameterized analysis tools grounded in standard libraries. It executes every candidate trace and retains a question only when its constraints determine a single trace. We then post-train a tool-using agent with a trace-aware reward that scores both the procedure and the answer. Experiments show that trace-aware reinforcement learning reaches near-perfect trace fidelity with half the tool calls of answer-only training, while improving answer accuracy. The trained agent generalizes to held-out workflow structures and tools and transfers zero-shot to external benchmarks. Compared with zero-shot frontier agents, it attains comparable answer accuracy with higher trace fidelity and fewer tool calls.

1 INTRODUCTION

Across signal-rich domains such as finance, healthcare, climate science, and industrial monitoring, analysts transform raw temporal observations into evidence for downstream decisions (Lim & Zohren, 2021; Liang et al., 2024). Such analysis typically proceeds through dependent steps: selecting an operation (*e.g.,* a Fourier transform or a stationarity test), binding it to a signal and time window, executing it, interpreting the intermediate result, and deciding what to analyze next. For example, assessing whether a sensor became unstable after an operational event may require identifying the event window, comparing pre- and post-event variability, and using the comparison to decide whether follow-up analyses are warranted. Such analyses currently demand expertise and effort, but language-model agents offer a promising path to automation. An agent can interpret natural-language questions and invoke tools that carry out the corresponding numerical procedures.

Executing these procedures reliably calls for tool-integrated reasoning (TIR), which makes each analytical step an explicit, exactly executed tool call (Yao et al., 2023; Schick et al., 2023; Qin et al., 2024). Without tools, the model must compute every quantity directly from the series, but LLMs cannot reliably perform such precise numerical computation (Gao et al., 2023; Bie et al., 2026). Moreover, a directly generated answer lacks interpretability because it reveals nothing about which operations were applied, with which parameters, or how intermediate results contributed to it, as in existing time-series QA benchmarks (Kong et al., 2025; Jing et al., 2026; Cai et al., 2024; Chen et al., 2025).

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.

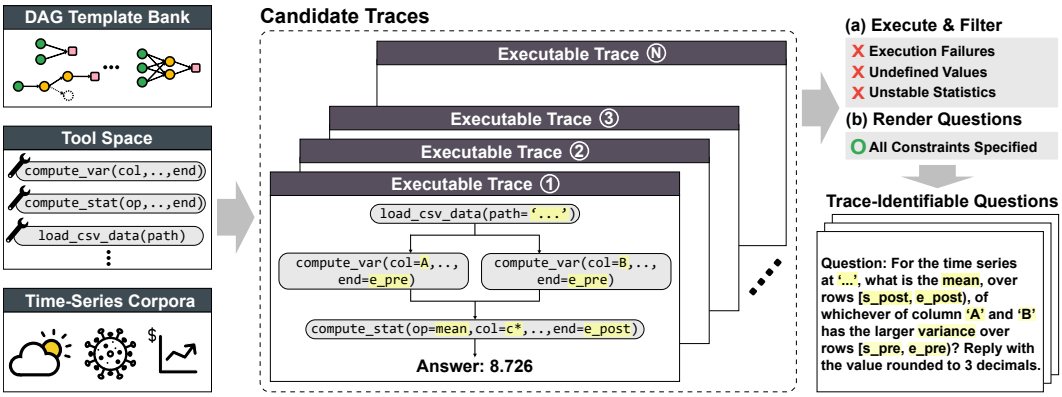


Figure 1: Trace-identifiable question construction in RITS. DAG templates, parameterized tools, and time-series corpora instantiate candidate executable traces, DAGs whose edges carry data dependencies and whose independent calls admit any execution order. Each candidate is executed and filtered for validity, then rendered into a natural-language question that is retained only when its stated constraints (highlighted) determine the trace within the modeled tool space.

Training and evaluating agents for such analyses, however, remains challenging, as the task requires two distinct abilities. First, when the question underspecifies the procedure, the agent must decide how to conduct the analysis. For instance, given the question "Did the sensor become unstable after the operational event?", the agent must choose among comparing pre- and post-event variance, testing for a change point at the event, and counting post-event anomalies. Once the procedure is determined, the agent must subsequently outline a concrete plan and execute the plan faithfully, calling the right tools with the right parameters and binding later calls to intermediate results.

When no single procedure is uniquely correct, supervising the choice lacks a ground truth. In the above question, for instance, any of the three procedures is a reasonable way to conduct the analysis. In contrast, when the intended procedure is uniquely determined, whether an agent executed it correctly can be checked exactly, because standard time-series analysis tools are well-defined. We therefore focus on supervising faithful execution in time-series analysis agents. This ability is demanding in time-series analysis, as procedures often involve complex, interdependent steps: they may compare independent measurements, follow data-dependent branches, or bind the arguments of later calls to earlier outputs. Moreover, a call with even one wrong parameter, such as a lag or window, computes the wrong quantity, yet returns a plausible number. Such mistakes cannot be caught from the final answer alone.

To train and evaluate agents against the intended procedure, we introduce **RITS**, a framework for multi-hop tool-integrated time-series reasoning. RITS grounds each question in a single executable *trace*, which specifies the required tool calls, their parameter bindings, and the dependencies among reasoning steps. Specifically, we define templates structured as directed acyclic graphs (DAGs), whose nodes are tool calls and whose edges encode dependencies between calls, adapting the bottom-up composition of MuSiQue (Trivedi et al., 2022). By assigning tools and parameter bindings to template nodes, we instantiate executable traces and render them into natural-language questions. A question is retained only when it is *trace-identifiable*: its stated constraints must determine the intended trace within the RITS tool space, ruling out ambiguity by construction (Figure 1). To cover a broad range of procedures and workflow structures, RITS combines a tool space of 308 parameterized time-series tools across 38 analysis categories with 101 DAG-based templates spanning two to five hops.

RITS uses the identifiable trace of each retained question as a reference for both training and evaluation of tool-using agents. We design a trace-aware reward that scores fidelity to the intended trace in addition to final-answer correctness, and post-train agents with group-relative policy optimization (GRPO) (Shao et al., 2024). At evaluation time, the trace serves as the reference both for scoring the executed workflow alongside the final answer and for localizing failures. We evaluate on an in-distribution split and on two controlled out-of-distribution splits that hold out entire DAG templates (*unseen-template*) or selected tools and parameter bindings (*unseen-tool*), and measure zero-shot performance on external time-series reasoning benchmarks. Trained on RITS, a 4B open-source

agent follows the intended procedure almost perfectly with fewer tool calls than an answer-only-trained agent, attains answer accuracy comparable to zero-shot frontier agents, and generalizes to held-out templates, tools, and external benchmarks.

Our contributions are summarized as follows:

- To evaluate executed time-series analyses against their intended procedures, we construct a **trace-identifiable** multi-hop QA dataset, where every question carries a unique executable reference trace.
- To train agents that follow the intended procedure of each question, we propose a **trace-aware reward** that scores an agent's executed trace against the reference, in addition to the final answer.
- We show that trace-aware training yields near-perfect trace fidelity with fewer tool calls than answer-only training, and that the trained agent attains answer accuracy comparable to zero-shot frontier agents and generalizes to held-out templates, tools, and external benchmarks.

2 Related Work

Tool-integrated LLM agents. Tool-integrated language agents extend LLMs beyond parametric text generation by interleaving reasoning with calls to external systems. ReAct couples natural-language reasoning with environment actions (Yao et al., 2023), while Toolformer induces API-call supervision from model-generated candidates (Schick et al., 2023). Subsequent systems scale tool use to large API spaces (Qin et al., 2024; Li et al., 2023), with dedicated benchmarks for tool selection and invocation quality (Chen et al., 2024; Patil et al., 2025). A complementary line uses executable programs or modular controllers to separate computation from language generation (Gao et al., 2023; Lu et al., 2023; Wang et al., 2024). Recent agentic training improves multi-step reasoning and tool use with execution feedback and reinforcement learning (RL) (Schulman et al., 2017; Shao et al., 2024; DeepSeek-AI, 2025; Gou et al., 2024; Singh et al., 2025; Zhang et al., 2025a), and process supervision extends rewards from final outcomes to intermediate steps (Uesato et al., 2022; Lightman et al., 2024). Large tool inventories also motivate sparse, dense, or diversity-aware retrieval over tool schemas to restrict the action space (Robertson & Zaragoza, 2009; Karpukhin et al., 2020; Zhang et al., 2025b; Carbonell & Goldstein, 1998).

Multi-hop QA. Multi-hop QA evaluates whether models can combine evidence across dependent steps. HotpotQA introduced bridge and comparison questions with supporting facts (Yang et al., 2018); 2WikiMultiHopQA combines structured knowledge-base paths with textual evidence (Ho et al., 2020); and MuSiQue constructs questions by composing connected single-hop questions to reduce shortcut solvability (Trivedi et al., 2022). Self-Ask elicits explicit sub-question decompositions (Press et al., 2023). More recent datasets increase the reasoning burden beyond extractive spans (Schnitzler et al., 2024; Park et al., 2026).

Time-series modeling and reasoning. LLMs and foundation models for time series have largely targeted forecasting (Gruver et al., 2023; Jin et al., 2024; Ansari et al., 2024; Woo et al., 2024; Das et al., 2024). Recent benchmarks instead probe time-series QA, multimodal temporal reasoning, and analysis assistants (Kong et al., 2025; Chen et al., 2025; Ye et al., 2025; Jing et al., 2026; Cai et al., 2024), most recently adding human-reviewed reasoning annotations and outcome-reward RL (Guan et al., 2026). They typically serialize the series as text or through a modality encoder and grade the final output. Appendix D compares RITS with these benchmarks.

Positioning of RITS. RITS connects tool-integrated agents, multi-hop QA, and time-series modeling by making the unique executable trace the unit of supervision and evaluation. Whereas text-based multi-hop QA links hops through textual evidence or intermediate answers, RITS links hops through computed outputs that bind the arguments of later tool calls. Whereas agentic training rewards outcomes or extends them with process supervision, RITS scores each executed workflow against the reference trace of its question.

3 THE RITS FRAMEWORK

In this section, we first formalize traces and trace identifiability (Section 3.1), then present the components of RITS: a tool space with explicit parameters (Section 3.2), DAG-based templates and trace-verified data construction (Sections 3.3–3.4), per-question candidate tool sets with retrieval (Section 3.5), and trace-aware policy optimization (Section 3.6).

3.1 TRACE-IDENTIFIABLE TIME-SERIES QA

A natural analysis question is often procedurally ambiguous. It leaves the choice of workflow open, and the answer can depend on that choice. An answer is then correct only relative to an intended workflow that the question does not state. RITS makes this intent explicit in the question. The stated constraints admit a unique workflow, which fixes the answer.

Let q be a natural-language question grounded in a time-series dataset $\mathcal{D}$, and let $\mathcal{A}$ denote a set of executable analysis tools. Each tool $a \in \mathcal{A}$ has an argument space Θ_a specifying choices such as the target signal (a column of $\mathcal{D}$), time window, or threshold. We represent a workflow as a *directed acyclic graph* (DAG)

$$\tau = (V, E, \ell, \vartheta), \tag{1}$$

whose nodes $v \in V$ are executable tool calls, each labeled with a tool $\ell(v) \in \mathcal{A}$ and arguments $\vartheta(v) \in \Theta_{\ell(v)}$, and whose edges $(u, v) \in E$ encode data dependencies: the output of u is consumed by v. We call this formal object a *trace*. Executing the trace computes

$$z_v = \mathrm{Exec}\big(\ell(v), \vartheta(v); \mathcal{D}, \{z_u\}_{u \in \mathrm{Pa}(v)}\big), \qquad y = g(\{z_v\}_{v \in S}), \tag{2}$$

where z_v is the output of call v, $\mathrm{Exec}(\cdot)$ runs the tool $\ell(v)$ with arguments $\vartheta(v)$ on $\mathcal{D}$ given the outputs of the parents $\mathrm{Pa}(v)$, and g is the aggregation function over $S \subseteq V$, the sink nodes designated by the template.

Because a call depends only on the outputs of its parents, any topological order yields the same intermediate values and final answer. Therefore, we call a question *trace-identifiable* when it admits a unique executable trace up to reordering of independent calls:

$$\mathcal{T}_{\mathcal{A}}(q, \mathcal{D}) \neq \varnothing \;\text{ and }\; \tau \sim \tau' \text{ for all } \tau, \tau' \in \mathcal{T}_{\mathcal{A}}(q, \mathcal{D}), \tag{3}$$

where $\mathcal{T}_{\mathcal{A}}(q, \mathcal{D})$ is the set of executable traces over $\mathcal{A}$ consistent with the constraints stated by q on $\mathcal{D}$, and $\tau \sim \tau'$ means the two traces have the same labeled nodes and the same dependency edges. RITS retains only questions that satisfy this condition, which is enforced by construction within the modeled tool space, as detailed in Appendix A.

3.2 LIBRARY-GROUNDED TOOL SPACE

To cover a realistic range of time-series analyses while keeping every trace verifiable, we construct a space of 308 parameterized tools across 38 analysis categories, grounded in standard numerical and statistical libraries such as NumPy and SciPy (Appendix B). The key design choice is to make every discriminating parameter an explicit argument. For example, a lag value or an anomaly threshold must be specified explicitly in the tool call rather than inherited from the defaults of the underlying library function. As a result, two traces with different parameter bindings remain distinct even when their final answers coincide.

A natural alternative is to replace the tool space with a single code-execution tool and let the model write each analysis as code (Gao et al., 2023; Wang et al., 2024). For trace-level evaluation, however, code offers no stable reference. The same analysis can be written in unboundedly many ways, so there is no canonical form to compare a produced analysis against. Moreover, parameters omitted from the code fall back to library defaults, so the executed analysis can deviate from the intended one. We therefore retain the parameterized tool space, where every reference trace has a canonical form and every parameter is explicit.

3.3 DAG-BASED QUESTION TEMPLATES

Many time-series analysis questions call for workflows richer than a linear chain. A workflow may select a signal based on an earlier measurement, combine the outcomes of independent tests into

a single conclusion, or follow a data-dependent branch. RITS captures these recurring workflow structures as reusable templates over the tool space, whose instantiations are traces. The template bank contains 101 DAG-based templates spanning two to five hops. Representative structures are sketched in Appendix C.

Formally, a template is a graph $G = (V, E)$ together with a node labeling function $\ell : V \to \mathcal{A}$, which fixes the tool called at each node, and argument functions

$$\vartheta(v) = \psi_v\big(q, \{z_u\}_{u \in \mathrm{Pa}(v)}\big) \in \Theta_{\ell(v)}, \qquad v \in V, \tag{4}$$

where ψ_v determines the arguments of call v from the question and the outputs of its parents, as in the example trace of Figure 1. Each template specifies both the computational structure and the natural-language constraints needed to make the reference trace identifiable.

3.4 Trace-Verified Data Construction

Dataset construction proceeds by instantiating DAG templates over real time-series corpora, binding each step to columns, windows, and parameter values, executing the resulting candidate traces, and retaining only examples that pass verification (Figure 1). A candidate is rejected when execution fails or yields a degenerate answer, such as an undefined statistic, or when the rendered question is not trace-identifiable. The retained dataset can be viewed as

$$\mathcal{S} = \big\{(q_i, \mathcal{D}_i, \tau_i, y_i) : \ \mathrm{Valid}(\tau_i, \mathcal{D}_i) \wedge \mathrm{Identifiable}(q_i, \mathcal{D}_i)\big\}, \tag{5}$$

where Valid holds when the trace executes to a well-defined answer, and $\mathrm{Identifiable}$ is the condition of Equation 3.

For each retained example, RITS stores the question, the final answer, the executable trace, and the intermediate answer of each step of the trace. The data are partitioned into in-distribution (ID) splits and two out-of-distribution (OOD) splits: unseen-template, which holds out entire DAG templates, and unseen-tool, which holds out selected tools and parameter bindings. Additional details are given in Appendix C.

3.5 Candidate Tool Sets and Tool Retrieval

Because the agent selects tools from the schemas in its prompt, including the full tool space would exceed a practical prompt budget. We therefore pair each question with a set of candidate tools. During training, the set contains the reference tools and distractors, capped at a fixed size. We call sets constructed this way oracle candidate sets. At evaluation time, we either construct the set as in training or obtain it from a dense retriever. Since we enforce identifiability against the full tool space, restricting the candidate set leaves the reference trace unchanged.

The retriever embeds a textual description of each tool, assembled from its name, category, parameter schema, and domain tags. It then ranks tools by dense similarity to the question. To cover the distinct tools that a multi-hop question requires, we re-rank this list with a redundancy penalty that discourages selecting closely similar tools. Because a question is phrased in natural language while tool descriptions consist of schema terms, a general-purpose text embedder can fail to place the reference tools among its top candidates. We therefore fine-tune the embedder contrastively on RITS questions, treating the reference tools of each question as positives and its distractor tools as negatives. Additional details are given in Appendix E.

3.6 Trace-Aware Policy Optimization

RITS trains an LLM policy that answers each question as a tool-using agent. At each step, it observes the question and previous tool outputs, chooses a tool with arguments, and receives the execution result. The trajectory ends when the policy returns a final answer or reaches its turn budget.

To guide the policy to reach the answer through the reference trace, RITS maximizes a trace-aware reward with GRPO (Shao et al., 2024). For a predicted trajectory $\hat{\tau}$ and answer $\hat{y}$, the reward is defined as

$$R(q, \hat{\tau}, \hat{y}) = \lambda_y r_y + \lambda_\tau r_\tau + \lambda_z r_z + \lambda_f r_f, \tag{6}$$

where $r_y(\hat{y}, y^*)$ scores the final answer; $r_\tau(\hat{\tau}, \tau^*)$ scores fidelity to the reference trace (Equation 7); r_z scores recovery of the intermediate values (Equation 8); and r_f scores the structural validity of the response format, the precondition for parsing the calls and the answer.

The trace term r_τ compares the predicted trajectory against the reference trace (Figure 2). It matches each node v of the reference trace to at most one predicted call c, and scores the agreement of each matched pair by $s(v, c) \in [0, 1]$ (Appendix F). We constrain the matching only by the dependency order of the reference DAG, so r_τ is invariant under exactly the reorderings that Equation 3 treats as identical. To aggregate the pair scores into r_τ, we let s_{tot} be the sum of s over the matched pairs, T the number of predicted calls, and $|V|$ the number of reference nodes. Then s_{tot}/T and $s_{\text{tot}}/|V|$ act as precision and recall. Since either alone would leave omitted or extra calls unpenalized, we take their harmonic mean and discount overly long trajectories:

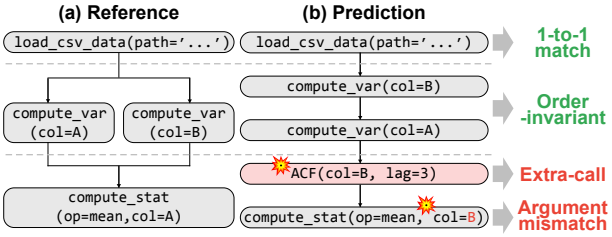


Figure 2: An example of trace-aware reward computation. Predicted calls are matched to reference nodes in dependency order, so reordering independent steps is not penalized while unmatched extra calls lower precision.

$$r_\tau = \frac{2\, s_{\text{tot}}}{T + |V|} \cdot e^{-\alpha \max(0,\, T - \kappa|V|)}. \tag{7}$$

The exponential factor leaves trajectories of up to $\kappa|V|$ calls unpenalized, so the policy can retry failed calls. Longer trajectories are discounted at rate α per excess call.

The intermediate term r_z rewards the recovery of the reference values themselves, since the trace term credits a matched call even when its computed value is wrong. Let Z be the intermediate values of the question. The intermediate term is then defined as

$$r_z = \frac{1}{|Z|} \sum_{z \in Z} \mathbb{1}[z \text{ appears in } \hat{\tau}]\,, \tag{8}$$

where z appears in $\hat{\tau}$ when its value is reproduced in any tool output or in the response text. Numerical values are compared under a small tolerance, the same rule used to grade the final answer. Additional details are given in Appendix F.

4 Experiments

The experiments address four research questions. **RQ1**: How does RITS perform against the baselines? **RQ2**: How much does each component, the reward terms and the tool retriever, contribute? **RQ3**: Does the trained policy generalize to unseen workflow structures, held-out tools, and external benchmarks? **RQ4**: Where do the remaining failures occur?

4.1 Experimental Setup

Datasets. All experiments use the RITS dataset, constructed as in Section 3.4. It contains multi-hop questions of two to five hops over 26 real series drawn from nine public corpora, split into 4,000 training questions, a validation split, and two test splits of 500 questions each, ID and OOD. The OOD split divides evenly into an *unseen-template* subset (held-out workflow structures) and an *unseen-tool* subset (held-out tools). Construction details are in Appendix C.

Baselines. We compare five baseline families. *Closed frontier and open-source models* run zero-shot with the same tools and agent loop as the RITS policy: GPT-5.6-sol (OpenAI, 2026), Gemini 3.1 Pro (Google DeepMind, 2026), and Claude Opus 4.8 (Anthropic, 2026); and Qwen3-4B (Yang & Qwen Team, 2025), Ministral-3-8B-Reasoning (Liu et al., 2026), and Qwen3.6-27B (Qwen Team, 2026). *Question-only prompting* answers from the question text alone, bounding what is achievable without tools or the series. *Code execution* replaces the tool space with a single Python tool. The base model delegates each computation to a frozen Qwen3-Coder-30B so that

Table 1: Comparison of the RITS policy with five baseline families on the test splits. SFT and RITS train the shared Qwen3-4B base. All tool-using systems share the splits and the five-turn budget, and those with the shared tool space receive retrieved candidate sets. Values are percentages except Calls. – denotes not applicable. Bold marks the best value in each column. †Excluded from the Calls comparison, as a single Python tool replaces the shared tool space. RITS values are the mean over three independent training runs (Appendix G.3).

Method	ID				OOD							
					Overall				Unseen template		Unseen tool	
	Acc.	Trace	OV-F1	Calls↓	Acc.	Trace	OV-F1	Calls↓	Acc.	Trace	Acc.	Trace
Closed frontier, tool use												
GPT-5.6-sol	52.2	92.8	87.0	5.37	67.4	92.7	87.8	5.40	58.8	92.7	76.0	92.8
Gemini 3.1 Pro	53.8	90.3	89.9	5.52	67.8	90.9	88.0	5.62	60.8	90.8	74.8	91.0
Claude Opus 4.8	64.4	91.0	90.1	5.64	**84.6**	90.0	89.1	5.41	**91.6**	87.2	77.6	92.7
Open-source, zero-shot tool use												
Qwen3-4B	37.0	40.4	18.6	18.93	34.4	44.9	33.2	13.37	38.8	49.5	30.0	40.2
Ministral-3-8B-Reasoning	45.2	78.4	75.9	5.48	55.8	83.6	79.9	5.58	56.0	86.5	55.6	80.7
Qwen3.6-27B	48.2	86.4	83.0	5.46	60.2	88.8	88.0	5.57	59.2	87.1	61.2	90.6
Question-only prompting												
GPT-5.6-sol, question only	24.0	–	–	–	33.4	–	–	–	24.8	–	42.0	–
Qwen3-4B, question only	21.6	–	–	–	19.6	–	–	–	14.4	–	24.8	–
Code execution, single Python tool												
Qwen3-4B + Qwen3-Coder-30B	50.2	–	–	2.74†	40.8	–	–	3.42†	30.0	–	51.6	–
Trained on reference traces												
SFT	53.8	88.0	81.3	5.14	63.2	**94.6**	87.8	5.26	54.8	92.2	71.6	**96.9**
RITS (ours)	**83.1**	**98.7**	**98.3**	**4.81**	69.5	94.0	**89.3**	**5.17**	60.8	**93.8**	**78.1**	94.1

coding ability does not bottleneck the comparison. *SFT* trains on multi-turn tool-use dialogues constructed from the reference traces, contrasting imitation with RL under the same trace supervision.

Implementation. The RITS policy post-trains Qwen3-4B with GRPO under the reward of Equation 6 and decodes in thinking mode. Evaluation uses retrieved top-20 candidate sets from the fine-tuned retriever. Run configurations and prompts are in Appendices G.2 and H.

Metrics. Answer correctness is scored as accuracy: normalized exact match for categorical answers and tolerance-based matching for numerical answers, with no partial credit. Trace fidelity is measured by two metrics. The trace score is the trace term r_τ against the reference trace, averaged over questions. Ordered-valid tool F1 (OV-F1) is reward-independent and scores tool-name micro-F1, counting a call only when it executes without error and follows the dependency order of the reference trace. We also report average tool calls (Calls), intermediate-hop accuracy (Interm.), the fraction of intermediate answers reproduced, and execution failures per question (Fail/q). Confidence intervals and grading details are in Appendix G.

4.2 RQ1: Performance against Baselines

Table 1 compares RITS with the five baseline families. On ID, RITS answers more questions correctly than every baseline, leading the closest frontier model by 18.7 percentage points. On OOD, only Claude Opus 4.8 answers more questions, reaching 84.6 accuracy against 69.5. The gap concentrates on unseen templates, and RITS stays slightly ahead on unseen tools. In contrast, RITS holds the highest trace score of all zero-shot systems on both splits, so RITS answers come from the intended procedure rather than from trial and error. OV-F1, which shares no quantity with the training reward, ranks RITS first on ID and essentially tied for first on OOD. Qwen3.6-27B, the strongest open-source model, remains well below RITS, reaching 60.2 on OOD. Overall, trace-aware RL turns a 4B open-source base into an agent that reaches answers through the intended procedure with fewer calls than any frontier model.

Moreover, four control experiments show that the gain requires RL, the tool space, and access to the series. (1) Without post-training, the shared Qwen3-4B base loses 46.1 percentage points (37.0

Table 2: Ablation study on the reward components of Equation 6. ✓ marks the active terms. All variants share the base model and training step and are scored on the test splits under oracle candidate sets. Values are percentages except Fail/q and Calls.

					ID					OOD				
Variant	r_y	r_f	r_τ	r_z	Acc.	Trace	Interm.	Fail/q	Calls	Acc.	Trace	Interm.	Fail/q	Calls
R1	✓	✓			76.4	35.4	88.1	4.06	8.99	79.4	31.7	87.4	4.39	9.63
R2	✓	✓	✓		84.8	99.0	94.6	0.03	4.82	82.4	98.9	95.4	0.00	5.10
R3	✓	✓		✓	85.8	99.0	96.0	0.04	4.80	83.4	97.3	94.3	0.03	5.08
RITS	✓	✓	✓	✓	84.8	99.4	96.3	0.02	4.79	84.4	98.2	95.0	0.00	5.09

against 83.1 on ID). (2) Without RL, SFT that imitates the same traces improves over the base but still trails RITS by 29.3 points on ID (53.8 against 83.1). Its trace score of 88.0 shows that imitation largely reproduces the reference trace yet fails to reach the answer. (3) Without the tool space, the code-execution baseline trails RITS by about 30 points even with a much stronger coding model. (4) Without the series, question-only prompting falls to chance level.

The frontier models trail RITS widely on ID and even score higher on OOD than on ID. Both patterns come mainly from the turn budget. On four- and five-hop ID questions they often use up all five turns without giving a final answer. With eight turns, GPT-5.6-sol becomes more accurate than the five-turn RITS policy, but its trace score drops sharply and its call count nearly doubles (Appendix G.1). The question-only scores may look high for a baseline without access to the series. The answer formats explain them. Numeric answers are almost never guessed correctly. Categorical questions, in contrast, state their two or three candidate answers in the question itself, such as 'higher' or 'lower'. Guessing among these accounts for the question-only score (Appendix G.2).

4.3 RQ2: Contribution of Each Component

Table 2 ablates the two procedure-level reward terms of Equation 6. Adding the trace term turns noisy tool use into faithful and efficient procedures. From R1 to R2 on ID, the trace score rises from 35.4 to 99.0. Execution failures fall to near zero, and tool calls drop by half. Answer accuracy improves on both splits as well. These gains are not an artifact of optimizing the trace score itself. Intermediate-hop accuracy, which the reward of R2 never scores, also rises to 94.6.

The intermediate term alone has a similar effect. R3 reaches a 99.0 trace score without ever scoring the trace. Reproducing the reference intermediate values requires calling the tools that compute them, so the intermediate reward supervises the procedure indirectly. Combining the two terms works best. RITS attains the best OOD accuracy of the four variants with the best ID trace score.

Table 3 compares the tool retriever that builds the candidate sets against two off-the-shelf alternatives, BM25 and a frozen embedder. The two alternatives find all reference tools for only 3.2% and 53.0% of OOD questions, respectively. We attribute this to the semantic gap between questions and tool descriptions. Questions are phrased in natural language, while the descriptions consist of schema terms. Contrastive fine-tuning on RITS questions closes this gap, reaching 96.6% on OOD and all ID questions.

4.4 RQ3: Generalization to Unseen Structures, Tools, and Benchmarks

Table 1 shows that unseen templates are harder than unseen tools, with accuracies of 60.8 and 78.1, respectively. Generalization is limited more by unfamiliar workflow structures than by unfamiliar tools. Figure 3 breaks the accuracy down by hop count. RITS generalizes reliably up to three hops, staying above 90% on both splits. Accuracy still degrades as questions demand more reasoning steps, with four- and five-hop questions accounting for most of the drop. In contrast, trace scores remain high at every hop. Even where the final answer fails, RITS executes a coherent workflow. Most of the remaining errors therefore arise at the last step, when RITS turns the computed results into a final answer. Appendix G.1 further analyzes the OOD accuracy drop.

Table 4 reports the zero-shot transfer of RITS to two external time-series reasoning benchmarks, TimeOmni-1 (Guan et al., 2026) and MTBench (Chen et al., 2025). RITS substantially improves

Table 3: Comparison of retrievers. R@k: recall of reference tools in the top k; Full@20: fraction of questions with all reference tools in the top 20; MRR@20: mean reciprocal rank of the best-ranked reference tool.

Split	Retriever	R@5	R@20	Full@20	MRR@20
ID	BM25	28.7	50.1	20.3	30.6
	Qwen3 frozen	67.9	83.5	54.0	75.6
	Qwen3 LoRA	**99.9**	**100.0**	**100.0**	**100.0**
OOD	BM25	21.2	32.6	3.2	22.2
	Qwen3 frozen	62.5	82.8	53.0	75.7
	Qwen3 LoRA	**87.7**	**98.9**	**96.6**	**95.1**

Table 4: Zero-shot transfer to external benchmarks: multiple-choice accuracy against the policy's pre-RL base (Qwen3-4B). Closed: GPT-4.1 for TimeOmni-1 and GPT-4o for MTBench, as reported in the original papers.

Benchmark	Task	Base	RITS	Closed
TimeOmni-1	Causality Discovery	30.5	**47.0**	28.7
	Scenario Understanding	61.0	53.5	**85.5**
	Decision Making	0.0	22.3	**25.5**
MTBench	Weather QA	32.0	**44.9**	41.7
	Finance QA	59.9	**78.6**	65.1

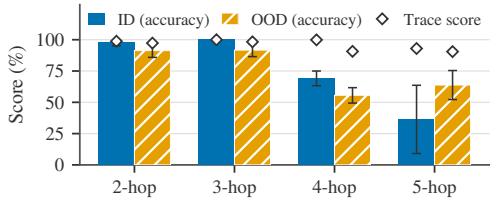


Figure 3: Answer accuracy by hop count with 95% CIs (ID under oracle sets, OOD under retrieved); diamonds mark trace scores, which stay high even where accuracy falls.

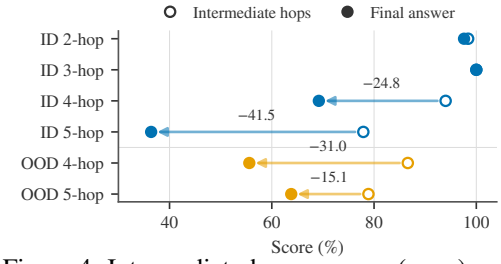


Figure 4: Intermediate-hop accuracy (open) versus final-answer accuracy (filled) by hop count. Labels give the drop from intermediate to final.

on its base on every task except scenario understanding. On three of the five tasks, RITS even exceeds the reported closed-model results. In particular, tasks that benefit from time-series analysis tools transfer well. Causality discovery, for example, improves by 16.5 points over the base. Appendix G.6 provides further task-level analysis.

4.5 RQ4: Failure Localization

Because RITS stores the reference trace and the intermediate answer of every hop for each question, failures can be localized to a specific stage of the workflow. Figure 4 compares intermediate-hop accuracy with final-answer accuracy by hop count. Here, ID is scored under oracle sets and OOD under retrieved sets. Most remaining failures arise after the analysis itself. On four- and five-hop ID questions, RITS computes 94.0% and 77.9% of the intermediate hops correctly, yet final accuracy falls to 69.2 and 36.4. This pattern indicates that the bottleneck is the reasoning step that combines tool results into a final answer. Answer accuracy alone would report all of these as undifferentiated failures. Appendix G provides further failure analyses.

5 Conclusion

We proposed RITS, which casts multi-hop time-series QA as trace-identifiable tool-integrated reasoning, scoring each analysis against a unique executable reference trace alongside the answer. The trace-aware reward yields near-perfect trace fidelity with half the tool calls of answer-only training while improving answer accuracy. The trained agent generalizes to held-out structures and tools, attains answer accuracy comparable to frontier agents with higher trace fidelity and fewer tool calls, and transfers zero-shot to external benchmarks. However, the identifiability guarantee of RITS holds only within the modeled tool space and template bank; open-world questions may admit other valid analyses. Systems outside this space, such as coding agents, remain comparable only by final answers. Also, template-generated questions may lack natural phrasing diversity, and the RITS-tuned retriever's near-perfect recall may not persist under varied wording. Potential directions for future research include broadening the range of supported tools and templates and extending trace evaluation to accommodate multiple valid analyses and diverse execution paradigms.

AI USE STATEMENT

In this work, we used generative AI tools to aid or polish writing, for retrieval and discovery (e.g., finding related work). We have not used generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, or interpret results, and providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, assisting with translation, and supporting qualitative and thematic data analysis are not applicable to this work. We have reviewed all AI-assisted work. All AI-assisted dataset components (e.g., candidate question templates and paraphrase lexicon entries) were subsequently passed through the programmatic generation, execution, and verification pipeline described in the paper: every retained question was executed against its reference trace, checked for trace identifiability, and re-executed to confirm the stored answer, and two authors manually inspected samples of the generated questions for correctness and naturalness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This study does not involve human participants, human-subject experimentation, or the collection of private or personally identifiable information. The time-series datasets used in our experiments are drawn from publicly available sources, with their provenance and construction process documented in the supplementary material. RITS is developed for research on the training and evaluation of tool-using agents. Although the analytical techniques considered in this work may be relevant to consequential domains such as finance, healthcare, and industrial monitoring, any deployment in such settings should be accompanied by appropriate human and domain-expert oversight and independent validation of the resulting analyses.

REPRODUCIBILITY STATEMENT

We document the components necessary to reproduce our experiments across the main paper and supplementary material. Appendix C details the dataset sources and construction process; Appendix B describes the tool registry and execution framework; Appendix E specifies the candidate-tool retrieval procedure; and Appendix F presents the reward formulation and training setup. Hardware and software configurations, hyperparameters, decoding settings, and evaluation procedures are reported in Appendix G.2, while Appendix H provides the prompts and a complete worked example. Results from three independent training runs are included in Appendix G.3. As summarized in Appendix I, we plan to publicly release the RITS dataset and reference traces, generation and verification pipeline, tool registry, training and evaluation code, scoring utilities, and trained policy checkpoints upon publication.

REFERENCES

Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, et al. Chronos: Learning the language of time series. *Transactions on Machine Learning Research (TMLR)*, 2024.

Anthropic. Introducing claude opus 4.8. https://www.anthropic.com/news/claude-opus-4-8, 2026.

Kinjal Basu, Ibrahim Abdelaziz, Kiran Kate, Mayank Agarwal, Maxwell Crouse, Yara Rizk, Kelsey Bradford, Asim Munawar, Sadhana Kumaravel, Saurabh Goyal, Xin Wang, Luis A. Lastras, and Pavan Kapanipathi. NESTFUL: A benchmark for evaluating LLMs on nested sequences of API calls. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 33538–33547, 2025.

Alex Bie, Travis Dick, Alex Kulesza, Prabhakar Raghavan, Vinod Raman, and Sergei Vassilvitskii. Ai-rithmetic, 2026. URL https://arxiv.org/abs/2602.10416.

Yifu Cai, Arjun Choudhry, Mononito Goswami, and Artur Dubrawski. TimeSeriesExam: A time series understanding exam, 2024.

Jaime Carbonell and Jade Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pp. 335–336. ACM, 1998.

Jialin Chen, Aosong Feng, Ziyu Zhao, Juan Garza, Gaukhar Nurbek, Cheng Qin, Ali Maatouk, Leandros Tassiulas, Yifeng Gao, and Rex Ying. MTBench: A multimodal time series benchmark for temporal reasoning and question answering, 2025.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, et al. T-Eval: Evaluating the tool utilization capability of large language models step by step. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

David F. Crouse. On implementing 2d rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4):1679–1696, 2016.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

DeepSeek-AI. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645:633–638, 2025.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023.

Google DeepMind. Gemini 3.1 Pro. `https://deepmind.google/models/model-cards/gemini-3-1-pro/`, 2026.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. ToRA: A tool-integrated reasoning agent for mathematical problem solving. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Nate Gruver, Marc Finzi, Shikai Qiu, and Andrew Gordon Wilson. Large language models are zero-shot time series forecasters. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Tong Guan, Zijie Meng, Dianqi Li, Shiyu Wang, Chao-Han Huck Yang, Qingsong Wen, Zuozhu Liu, Sabato Marco Siniscalchi, et al. TimeOmni-1: Incentivizing complex reasoning with time series in large language models. In *International Conference on Learning Representations (ICLR)*, 2026.

Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics (COLING)*, 2020.

Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y. Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, and Qingsong Wen. Time-LLM: Time series forecasting by reprogramming large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

Baoyu Jing, Sanhorn Chen, Lecheng Zheng, Boyu Liu, Zihao Li, Jiaru Zou, Tianxin Wei, Zhining Liu, Zhichen Zeng, Ruizhong Qiu, Xiao Lin, Yuchen Yan, Dongqi Fu, Jingchao Ni, Jingrui He, and Hanghang Tong. TSAQA: Time series analysis question and answering benchmark. In *Proceedings of the GEM Workshop at ACL 2026*, 2026.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

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 Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025.

Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long- and short-term temporal patterns with deep neural networks. In *Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2018.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Yuxuan Liang, Haomin Wen, Yuqi Nie, Yushan Jiang, Ming Jin, Dongjin Song, Shirui Pan, and Qingsong Wen. Foundation models for time series analysis: A tutorial and survey. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2024.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*, 2024.

Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379 (2194):20200209, 2021.

Alexander H. Liu, Kartik Khandelwal, Sandeep Subramanian, Victor Jouault, Abhinav Rastogi, Adrien Sadé, Alan Jeffares, Albert Jiang, et al. Ministral 3, 2026. URL https://arxiv.org/abs/2601.08584.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

OpenAI. Gpt-5.6: Frontier intelligence that scales with your ambition. https://openai.com/index/gpt-5-6/, 2026.

Sungho Park, Jueun Kim, and Wook-Shin Han. SPARTA: Scalable and principled benchmark of tree-structured multi-hop QA over text and tables. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=8KE9qvKhM4.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=2GmDdhBdDk.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations (ICLR)*, 2024.

Qwen Team. Qwen3.6-Plus: Towards real world agents. https://qwen.ai/blog?id=qwen3.6, April 2026.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessí, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: language models can teach themselves to use tools. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, 2023.

Julian Schnitzler, Xanh Ho, Jiahao Huang, Florian Boudin, Saku Sugawara, and Akiko Aizawa. MoreHopQA: More than multi-hop reasoning, 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024.

Joykirat Singh, Raghav Magazine, Yash Pandya, and Akshay Nambi. Agentic reasoning and tool integration for LLMs via reinforcement learning, 2025.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics (TACL)*, 2022.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback, 2022.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, et al. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17:261–272, 2020.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. Unified training of universal time series forecasting transformers. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In *Advances in Neural Information Processing Systems*, 2021.

Mengsong Wu, Tong Zhu, Han Han, Chuanyuan Tan, Xiang Zhang, and Wenliang Chen. Seal-tools: Self-instruct tool learning dataset for agent tuning and detailed benchmark. In *CCF International Conference on Natural Language Processing and Chinese Computing (NLPCC)*, 2024.

An Yang and Qwen Team. Qwen3 technical report, 2025.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Wen Ye, Jinbo Liu, Defu Cao, Wei Yang, and Yan Liu. When LLM meets time series: Can LLMs perform multi-step time series reasoning and inference, 2025.

Yabo Zhang, Yihan Zeng, Qingyun Li, Zhen Hu, Kavin Han, and Wangmeng Zuo. Tool-R1: Sample-efficient reinforcement learning for agentic tool use, 2025a.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models, 2025b. URL https://arxiv.org/abs/2506.05176.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

Luca Zhou, Pratham Yashwante, Marshall Fisher, Alessio Sampieri, Zihao Zhou, Fabio Galasso, and Rose Yu. CaTS-Bench: Can language models describe time series? In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 34479–34519, 2026.

Supplementary Materials for "RITS: Reasoning with Identifiable Traces for Time-Series Analysis Agents"

A Trace Equivalence and Identifiability

Partial-order equivalence. We compare traces up to *partial-order equivalence*: $\tau \sim \tau'$ iff they have the same labeled nodes and the same dependency edges, i.e. they differ only by reordering independent steps. The equivalence only defines when two traces count as the same trace. Whether a question admits exactly one such trace is the separate identifiability condition below.

Canonical serialization. Each DAG is serialized in a deterministic topological order. Nodes are sorted by dependency level, and nodes within a level are ordered by the canonical JSON encoding of $(\ell(v), \vartheta(v))$. Equivalent DAGs map to the same string, so orderings of independent steps are never counted as distinct traces.

Identifiability condition. A question is trace-identifiable when Equation 3 of the main text holds. Equivalently, $\left| \mathcal{T}_{\mathcal{A}}(q, \mathcal{D}) / \sim \right| = 1$, where $\mathcal{T}_{\mathcal{A}}(q, \mathcal{D})$ is the set of executable DAGs over $\mathcal{A}$ consistent with q on $\mathcal{D}$. We do not claim this as a global theorem over all conceivable analyses. It is *enforced by construction* at two scoped levels.

(i) *Parameter identification:* for every node, each discriminating argument, such as a test variant, lag, threshold, or window, must be witnessed in the question, either verbatim or through one of the descriptive surface forms declared in the paraphrase lexicon. A rendered question that fails to witness any argument of any node is rejected. (ii) *Within-tool distinctness:* parametric variants are not separate tools but different argument bindings of the same tool. Computing the mean and computing the median, for instance, are two variants of one statistic tool that differ in a single argument. We construct the paraphrase lexicon so that the phrasings admitted for different values of the same argument never overlap, and reverse-lookup uniqueness tests enforce this separation. No admitted phrasing therefore describes both the mean and the median variant, and a question that witnesses the intended binding cannot simultaneously satisfy a sibling.

Questions reference tools only through tool-specific paraphrases, and a name-leakage filter rejects questions containing raw tool names. Candidates with degenerate execution (undefined statistics, tied comparisons, numerically unstable answers) are rejected. The guarantee is thus relative to the tool space, the template bank, and the paraphrase lexicon.

B Implementation Details

Tool registry. The registry contains 308 entries across 38 categories. The tools are grounded in standard numerical and statistical libraries such as NumPy, SciPy, statsmodels, scikit-learn, and PyWavelets, and span descriptive, correlation, spectral and wavelet, anomaly, change-point, hypothesis-testing, and complexity analyses. Each entry records the tool name, category, output type, input arity, parameter schema, finite-valued parameter slots when applicable, and domain tags used for retrieval and filtering.

Schema conversion and execution. Runtime tool schemas are generated directly from the registry, with no separate static configuration. Each tool call is dispatched to the shared time-series toolkit instance for that trajectory, and outputs are converted into JSON-safe values before being returned to the model. Invalid tool names, malformed arguments, and runtime exceptions are recorded as tool-execution failures.

Per-trajectory state. Each rollout owns a trajectory-scoped time-series state. The data-loading call materializes the relevant file and columns, and subsequent analytical calls operate on that shared

state without mutating it. This read-only design allows independent steps to be executed in any relative order and mirrors an interactive analysis session while preventing state leakage across questions.

C Dataset Construction Details

Generation pipeline. The generator instantiates DAG templates over registered time-series corpora, executes candidate traces, rejects invalid or ambiguous candidates, deduplicates retained traces, partitions examples into ID and OOD pools, and writes the resulting splits. The output consists of train, validation, in-distribution test, and OOD test splits.

Corpora. The domain registry provides 26 real series drawn from nine public collections: the four ETT transformer-load series (Zhou et al., 2021), the electricity-consumption and exchange-rate series prepared by Lai et al. (2018), the weather and influenza-like-illness series of the long-horizon forecasting suite (Wu et al., 2021), three traffic series of PEMS08 (Guo et al., 2019), twelve equity series of individual companies, a market index and volatility pair, and a daily climate record, the latter three collected from public market and climate archives. Sampling cadences range from five minutes to weekly, and each series carries named row windows, such as quarters, months, and halves, that templates bind when instantiating questions.

Template metadata. The template bank contains 101 templates. Each template declares its category, semantic hop count, expected executable-call count, answer type, rejection rules, and domain constraints. Hop counts are semantic: a hop is one reasoning step that the question chains, so a selection over two candidate signals counts as one hop even though it executes two tool calls, and the initial data-loading call counts toward no hop. All reported hop counts follow this convention, and the executable-call count of a template can therefore exceed its hop count. Figure 5 sketches representative template structures beyond the bridge of Figure 1. A comparison reduces two independent measurements; a bridge computes a statistic on the signal selected by earlier measurements; a consensus merges the outcomes of independent tests; a threshold chain binds an intermediate output as the argument ϑ of the next call; a conditional branch executes only the branch selected by an intermediate result; and a four-hop tiebreak compares three candidate signals and breaks the tie between the top two with a second statistic. These shapes correspond to the template categories analyzed in Appendix G.4.

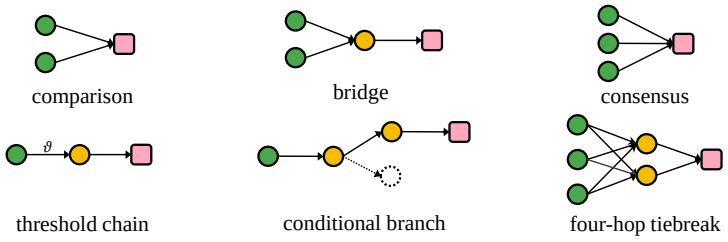


Figure 5: Representative template structures. Filled circles are tool calls, the square is the aggregation function g, edges carry outputs consumed by later steps, and the dashed node marks a branch that is not selected.

OOD construction. Unseen-template OOD examples are produced by holding out entire DAG templates from training. Unseen-tool OOD examples are produced by holding out selected tools and specific parameter bindings while other bindings of the same tools remain in training.

D Comparison with Time-Series QA Benchmarks

Table 5 situates RITS against recent time-series QA benchmarks (Kong et al., 2025; Chen et al., 2025; Jing et al., 2026; Cai et al., 2024; Guan et al., 2026; Zhou et al., 2026) along the dimensions that motivate this work. Existing benchmarks grade final outputs. TimeOmni-1 (Guan et al., 2026) is the closest design: it additionally provides human-reviewed reasoning annotations, in-distribution

and out-of-distribution testbeds, and RL with an outcome-level reward. The distinction RITS adds is the reference itself: an executable and unique tool-call trace per question, which upgrades reasoning annotations to a verifiable procedure-level reward and replaces domain-transfer generalization with controlled template and tool holdouts. No surveyed benchmark provides executable reference traces or a uniqueness guarantee. TimeSeriesExam (Cai et al., 2024) controls question quality through item response theory, which certifies answer clarity and leaves workflow identity unconstrained, and CaTS-Bench (Zhou et al., 2026) uses program-aided baselines without treating programs as ground truth. On scale, RITS trades corpus size for per-question guarantees: every released question passes execution, uniqueness, and re-execution checks, and the 500-question test splits support the confidence intervals of Appendix G.2.

Table 5: Positioning against time-series QA benchmarks. Step annotations: reasoning-step annotations shipped with examples. OOD axes: controlled generalization splits beyond a random test split. ✗ marks an absent property.

Benchmark	Series data	Scale	Step annotations	OOD axes
Time-MQA (Kong et al., 2025)	real + synthetic, 12 dom.	∼200k	✗	✗
MTBench (Chen et al., 2025)	real, 2 domains	22k	✗	✗
TSAQA (Jing et al., 2026)	real, 13 domains	210k	✗	✗
TimeSeriesExam (Cai et al., 2024)	synthetic, 104 templates	700+	✗	✗
TimeOmni-1 (Guan et al., 2026)	real, 10 domains	23.6k	human-reviewed CoT	domain transfer
CaTS-Bench (Zhou et al., 2026)	real, 11 domains	1.7k + 20k synth.	✗	✗
RITS	real, 26 series	5k	unique executable trace	template / tool

E Retriever and Candidate-Set Details

RITS restricts the set of tools shown to the model for a given question, drawing candidates from an oracle construction or from a dense retriever. Oracle sets isolate tool-use behavior from retrieval failure.

Candidate-set construction. Oracle sets contain the reference tools of the question and same-domain distractor tools, capped at a fixed size of 20 by default. Retrieved sets are the top-k tools returned by the retriever for the question text. In both modes the data-loading tool is force-included by the tool loader, and the retriever therefore excludes it from its own top-k.

Retriever. The retriever embeds one synthetic document per registry entry. Each document is generated from the entry's name, category, parameter schema, and domain tags, and is stale-checked against a registry hash so the index is rebuilt whenever the registry changes. Retrieval scores dense similarity between the question embedding and the document embeddings, followed by a family-aware maximal-marginal-relevance re-rank that promotes coverage of distinct tool families among the top-k. Concretely, retrieval forms a candidate pool $\mathcal{C}$ containing the $4k$ descriptions most similar to the question and then greedily selects k tools. With S the set selected so far, each step adds

$$
\begin{aligned}
d^\star = \operatorname*{arg\,max}_{\substack{d \in \mathcal{C} \setminus S \\ n_S(\mathrm{fam}(d)) < c}} \big[\, \lambda\, s(q,d) - (1-\lambda)\, \rho(d,S) \,\big], \\
\rho(d,S) = \begin{cases} 1 & \text{if } \mathrm{fam}(d) \in \mathrm{fam}(S), \\ \max_{d' \in S} s(d,d') & \text{otherwise,} \end{cases}
\end{aligned}
\tag{9}
$$

where $s(\cdot,\cdot)$ is cosine similarity in the embedding space, $\mathrm{fam}(d)$ is the tool family of d, $n_S(f)$ counts the selected members of family f, $\rho(d,\varnothing) = 0$, and the defaults are the per-family cap $c = 2$ and trade-off $\lambda = 0.7$. A candidate whose family is already represented in S incurs the maximum redundancy penalty, and other near-duplicates are penalized by their similarity to already-selected descriptions. In the current registry every entry constitutes its own family, so the per-family cap is inactive and diversification is driven by the redundancy penalty ρ. The encoder is a Qwen3-Embedding-8B model, evaluated frozen and after parameter-efficient contrastive fine-tuning (LoRA) with a multi-positive InfoNCE objective in which all reference tools of a question serve as positives.

The negative pool combines the question's distractor tools with the positives of the other questions in the batch.

F Training and Reward Details

Agent loop. Training uses a multi-turn tool-use loop in which the policy emits structured tool calls, receives execution outputs as observations, and then either continues the analysis or emits a final answer. Tool-output tokens are treated as observations and are excluded from the policy loss.

Reward components. The reward combines answer correctness, output-format validity, trace fidelity, and intermediate-step agreement. Trace fidelity includes tool matching, parameter matching, trace precision and recall, length penalties, and optional execution-output agreement. The default weights are 0.55 (answer correctness), 0.25 (trace fidelity), 0.05 (intermediate agreement), and 0.20 (format validity), normalized over the components active in a given run. The ablation of Section 4.3 toggles the trace and intermediate terms.

Trace-fidelity term. In Equation 7, the reference trace τ^* has nodes V partitioned into dependency levels $L_1, \ldots, L_m$, and the predicted trajectory is the call sequence $c_1, \ldots, c_T$. Each node is one executable tool call of the reference trace, so $|V|$ includes the initial data-loading call and can exceed the hop count of the template (Appendix C). Nodes count the calls a workflow comprises, while levels are the topological layers of its DAG: nodes within a level are mutually independent, and a node in a later level consumes outputs of earlier levels. The pair score is

$$\begin{aligned} s(v,c) &= \max\big(s_{\text{sig}}(v,c),\, s_{\text{out}}(v,c)\big), \\ s_{\text{sig}}(v,c) &= \frac{1}{2}\,\mathbb{1}\big[\text{name}(c) = \ell(v)\big] + \frac{1}{2}\cdot\frac{\big|\{\, a \in A_v : c \text{ reproduces } \vartheta_a(v)\,\}\big|}{|A_v|}, \end{aligned} \tag{10}$$

where A_v is the set of reference arguments of node v (the ratio is 1 when A_v is empty) and extra arguments of c are not penalized. The output score $s_{\text{out}}(v,c)$ is 1 when the executions of v and c both succeed and their outputs match under the answer tolerance, and 0 otherwise. The maximum grants full credit to a call that reproduces the reference output even when its signature differs. Algorithm 1 gives the matching procedure: levels are processed in order, each level solves an exact maximum-weight assignment between its nodes and the eligible calls, and the cursor keeps eligibility order-monotone, so later levels may only match calls that occur after the last call matched by earlier levels. Each per-level assignment is solved exactly with `linear_sum_assignment` from SciPy (Virtanen et al., 2020), an implementation of the modified Jonker–Volgenant algorithm of Crouse (2016) for rectangular cost matrices. Pair scores are negated to cast the maximization as a minimum-cost assignment, and pairs with $s(v, c_i) = 0$ are dropped after the solve (the discard step of Algorithm 1). The penalty rate and call threshold of Equation 7 are $\alpha = 0.5$ and $\kappa = 1.5$.

Optimization. The current training path uses group-relative policy optimization (Shao et al., 2024). All reinforcement-learning runs share one configuration: Qwen3-4B, rollout group size 3, batch size 64, learning rate 10^{-6}, KL coefficient 0.001, five assistant turns, and maximum generation length 4096, trained for 10 epochs to step 620 with rollouts sampled at temperature 0.7 and top-p 0.9 in thinking mode. Evaluation decodes greedily. SFT fine-tunes the full model for 10 epochs with global batch size 64 and learning rate 5×10^{-6} under a cosine schedule, selecting the checkpoint with the lowest validation loss, on multi-turn dialogues that re-execute each reference trace with executed tool responses and the same retrieved top-20 tool menus as evaluation. The SFT policy is served with per-turn reasoning retained in context, matching its training format, as is the case for RITS rollouts. Zero-shot baselines use standard reasoning-stripped serving.

Algorithm 1 DAG-aware trace matching for Equation 7

Require: reference nodes V in levels $L_1, \ldots, L_m$; predicted calls $c_1, \ldots, c_T$; pair score $s(v, c) \in [0, 1]$
Ensure: total matched score s_{tot}

$s_{\text{tot}} \leftarrow 0$; $cursor \leftarrow 1$; $U \leftarrow \emptyset$ ▷ U: positions of matched calls
for $j = 1$ **to** m **do**
  $C \leftarrow \{\, i : i \geq cursor,\ i \notin U \,\}$ ▷ eligible call positions
  $M_j \leftarrow \arg\max_M \sum_{(v,i)\in M} s(v, c_i)$ over one-to-one $M \subseteq L_j \times C$ ▷ maximum-weight assignment
  discard pairs of M_j with $s(v, c_i) = 0$
  for $(v, i) \in M_j$ **do**
    $s_{\text{tot}} \leftarrow s_{\text{tot}} + s(v, c_i)$; $U \leftarrow U \cup \{i\}$
  end for
  if $M_j \neq \emptyset$ **then**
    $cursor \leftarrow \max\{\, i : (v, i) \in M_j \,\} + 1$
  end if
end for
return s_{tot}

G Supplementary Results and Analyses

Unless stated otherwise, the analyses use the original evaluation runs: the ID side scored on the held-out validation split under oracle candidate sets, and the OOD split under retrieved top-20 sets. ID figures in this appendix therefore differ from the ID test figures of Table 1 by sampling variation. Analyses of a single policy use Run 3 of Appendix G.3.

G.1 Inference-Time Sensitivity

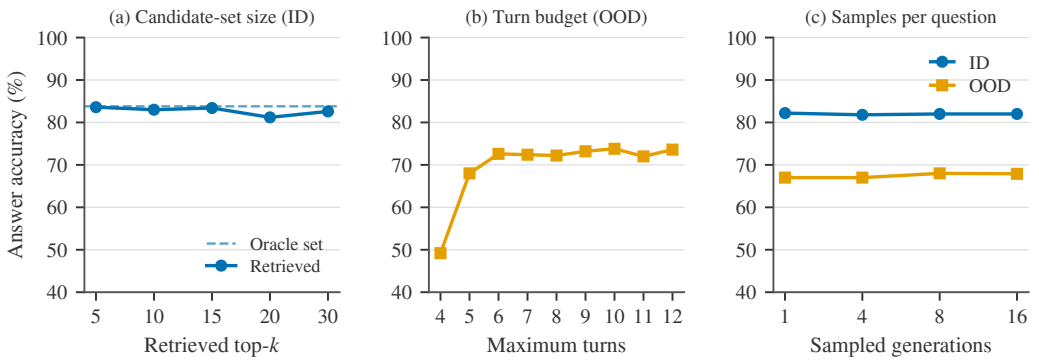


Figure 6: Inference-time sensitivity of answer accuracy. (a) Candidate-set size on ID: retrieved top-k against the oracle set (dashed). (b) Maximum turn budget on OOD. (c) Averaging over N sampled generations per question. Only the turn budget has a substantial effect. Exact values are in Tables 8, 9, and 11.

Figure 6 summarizes three inference-time sweeps. The turn-budget and sampling sweeps decode under retrieved top-15 candidate sets. Answer accuracy on ID stays within 81.2 to 83.8 as the candidate set grows from the oracle set to retrieved top-30. A compact candidate set is therefore sufficient.

The turn budget is the only setting with a substantial effect, and the dense sweep of Table 9 localizes it. OOD answer accuracy rises from 49.2 at four turns to 68.0 at five and 72.6 at six, and every budget from six through twelve stays within 72.0–73.8 without a monotone trend. The shared five-turn budget of Table 1 therefore sits at the base of this plateau, and budgets beyond six turns add at

most 1.2 points. Turns bound answers rather than tool calls. Four- and five-hop workflows average 6.4 to 6.6 calls, but independent calls are issued in parallel within a turn, and both the call count (5.15–5.23) and the trace score (93.4–94.1) are flat across the entire sweep, including the four-turn budget. At small budgets, the truncation therefore falls on the final answer turn, consistent with RQ4. Averaging over more sampled generations changes accuracy by at most one point on either split, so sampling diversity does not recover these failures either. The rows of Table 11 report direct means over all samples. Table 10 ablates the thinking-mode decoding of Section 4.1.

Table 6 compares the policy under its two candidate-set conditions on the full metric set, scored on the test splits of Table 1. Oracle sets, the training condition, contain the reference tools and same-domain distractors capped at 20. Retrieved sets are the deployment condition of Table 1. The swap moves ID answer accuracy by 2.0 points and OOD answer accuracy by 14.2, while question novelty itself costs little. Under oracle sets, OOD accuracy is only 0.4 points below ID. Retrieval recall does not explain the OOD shift, since the fine-tuned retriever reaches near-ceiling recall (Table 3), and neither does set size, which moves accuracy by at most 2.6 points in the sweeps below. What changes is the composition of the distractors. Oracle sets draw them from the same domain as the question, retrieved sets from the whole tool space, and the swap hurts mainly on novel questions.

Table 6: The RITS policy under retrieved and oracle candidate sets. Columns as in Table 1; values are percentages except Calls.

Candidate sets	ID				OOD							
					Overall				Unseen template		Unseen tool	
	Acc.	Trace	OV-F1	Calls	Acc.	Trace	OV-F1	Calls	Acc.	Trace	Acc.	Trace
Retrieved top-20	82.8	98.9	98.1	4.81	70.2	93.4	89.7	5.30	62.0	93.5	78.4	93.2
Oracle	84.8	99.4	99.3	4.79	84.4	98.2	97.9	5.09	84.4	97.7	84.4	98.6

The turn budget also bounds the frontier baselines, in the opposite direction. Table 7 extends the budget of GPT-5.6-sol from the shared five turns to eight. At five turns GPT returns no final answer on 232 ID and 95 OOD questions, almost all four- and five-hop. This larger unanswered mass on ID explains why the frontier rows of Table 1 score higher on OOD than on ID. Three added turns convert most of these into answers, 94% of them correct on ID, and lift GPT to 84.8 ID and 79.0 OOD answer accuracy, above the five-turn RITS policy on both splits. The added budget is spent searching outside the reference trace rather than completing it. The trace score falls from 92.8 to 73.4 on ID and from 92.7 to 83.4 on OOD, execution failures rise, and calls grow to about eight per question. OV-F1, which shares no quantity with the reward, falls in step, from 87.0 to 67.5 on ID and from 87.8 to 70.6 on OOD. The answers GPT produces only in the added turns carry trace scores of 52.4 on ID and 28.6 on OOD at 13.8 and 20.3 calls per question. The RITS policy, in contrast, completes its workflows within the shared budget. The same extension moves its OOD accuracy only from 68.0 to 72.2 at five calls per question throughout (Table 9). A relaxed budget therefore lets GPT-5.6-sol overtake the five-turn policy's answer accuracy by broad search, at a cost that the trace, call, and failure metrics record. Table 1 compares systems on reaching the answer through the reference trace within the same interaction budget.

Table 7: GPT-5.6-sol under the shared five-turn budget and an extended eight-turn budget, against the five-turn RITS policy under retrieved sets (Run 3 of Appendix G.3). Values are percentages except Fail/q and average calls.

	ID					OOD				
Method	Acc.	Trace	OV-F1	Fail/q	Calls	Acc.	Trace	OV-F1	Fail/q	Calls
GPT-5.6-sol, five turns	52.2	92.8	87.0	0.46	5.37	67.4	92.7	87.8	0.22	5.40
GPT-5.6-sol, eight turns	84.8	73.4	67.5	0.74	8.31	79.0	83.4	70.6	0.33	8.03
RITS, five turns	82.8	98.9	98.1	0.03	4.81	70.2	93.4	89.7	0.15	5.30

Table 8: Policy inference across candidate-set sources and sizes on the ID split, as direct means over the evaluation outputs.

Split	Candidate set	n	Answer Acc.	Trace score	Fail/q	Avg. calls
ID	Oracle	500	83.8	99.6	0.01	4.78
ID	k=5	500	83.6	99.4	0.02	4.79
ID	k=10	500	83.0	98.7	0.03	4.81
ID	k=15	500	83.4	99.1	0.03	4.81
ID	k=20	500	81.2	99.0	0.03	4.81
ID	k=30	500	82.6	99.0	0.03	4.80

Table 9: OOD turn-budget sensitivity. Values are percentages except Fail/q and average calls.

Max turns	n	Answer Acc.	Trace score	Fail/q	Avg. calls
4	500	49.2	93.4	0.10	5.15
5	500	68.0	94.1	0.11	5.23
6	500	72.6	93.8	0.13	5.22
7	500	72.4	93.9	0.10	5.20
8	500	72.2	93.6	0.11	5.19
9	500	73.2	93.5	0.13	5.23
10	500	73.8	93.6	0.12	5.21
11	500	72.0	93.7	0.11	5.21
12	500	73.6	93.8	0.10	5.22

Table 10: Thinking versus non-thinking policy inference under oracle candidate sets. n denotes questions per split; values may differ from Table 6 by up to 1.2 points of run-to-run variation across evaluation jobs.

Split	Mode	n	Acc.	Trace score	Fail/q	Avg. calls
ID	Non-thinking	500	45.8	93.4	0.03	4.11
ID	Thinking	500	**84.4**	**99.6**	0.01	4.79
OOD	Non-thinking	500	41.4	89.9	0.01	4.20
OOD	Thinking	500	**83.2**	**98.3**	0.00	5.10

Table 11: Raw N-sample average metrics.

Split	N	Sample rows	Answer Acc.	Trace score	Fail/q	Avg. calls
ID	1	500	82.2	99.4	0.02	4.77
ID	4	2000	81.8	99.1	0.02	4.78
ID	8	4000	82.0	99.1	0.02	4.78
ID	16	8000	82.0	99.2	0.03	4.78
OOD	1	500	67.0	93.0	0.10	5.06
OOD	4	2000	67.0	93.4	0.08	5.08
OOD	8	4000	68.0	93.9	0.08	5.11
OOD	16	8000	67.9	93.6	0.09	5.09

G.2 Run Configuration and Confidence Intervals

All tool-using systems receive the same per-question candidate sets, turn budget, and scoring. The per-run configurations are listed below, and prompts are provided in Appendix H. A direct-prompting comparison that places the raw series in context is omitted because the series exceed practical context lengths.

All experiments were conducted on single nodes of a SLURM cluster (Rocky Linux 9.4) with four NVIDIA H200 141 GB GPUs, under Python 3.12, PyTorch 2.8, verl 0.8.0 with vLLM 0.11 rollouts, Transformers 4.57, and Ray 2.51, with tool computations running on NumPy 2.2, SciPy 1.16, statsmodels 0.14, and scikit-learn 1.7. Training runs use random seeds, and rollout sampling on the serving engine is nondeterministic across executions, so replication targets the run-to-run band of Appendix G.3 rather than bitwise identity.

Confidence intervals are 95% percentile bootstrap over questions with 1,000 resamples. Reference-trace re-execution, which runs the reference trace verbatim through the same pipeline, calibrates the answer metric. Re-execution reaches 100.0 accuracy on every split with no execution failures, so the answer-matching step contributes no measurable error and residual gaps are attributable to the models. Grading details: numerical answers match under relative and absolute tolerance 10^{-6}, the same grader scores training rewards and evaluation, and a single assistant turn may issue several tool calls, so call counts can exceed the turn budget. An answer counts only when it is emitted outside the model's reasoning block. An answer that appears only inside the `think` block is scored as incorrect, the same output discipline that the format reward enforces for RITS. The intervals in Table 14 summarize sample-level uncertainty over questions, computed for the three-run mean reported in Table 1. Run-to-run variation across the three RITS training runs is reported in Appendix G.3.

Question-only prompting calibrates the opposite end of the answer metric: how far it can be pushed without observing the series. Table 12 decomposes the question-only rows of Table 1 by answer type. Numeric answers are essentially never produced without computation, at 1.0 to 6.8 accuracy against the 10^{-6} tolerance, and the few hits are values the question alone determines, such as the quantile rank of a median, 0.5 by definition, or counts that coincide with the stated window length. The question-only score therefore concentrates in categorical answers, whose two- or three-option label sets the question itself states, and sits at the level such label sets allow by guessing.

Table 12: Question-only answer accuracy decomposed by answer type. Categorical answers are drawn from the label sets stated in the question.

Model	Split	n	Acc.	Numeric n	Numeric Acc.	Categorical n	Categorical Acc.
GPT-5.6-sol	ID	500	24.0	203	1.0	297	39.7
GPT-5.6-sol	OOD	500	33.4	207	6.8	293	52.2
Qwen3-4B	ID	500	21.6	203	1.0	297	35.7
Qwen3-4B	OOD	500	19.6	207	4.3	293	30.4

Table 13: Per-run training configuration. Candidate sets and decoding refer to training rollouts. Evaluation decodes greedily, and its candidate-set conditions are stated with each experiment.

Run	Model	Candidate set	Optimizer	Group size	Max turns	Decoding
RITS main (full reward)	Qwen3-4B	GT+distractor, cap 20	GRPO	3	5	sampled, 0.7 / 0.9
Reward variants R1–R3	Qwen3-4B	GT+distractor, cap 20	GRPO	3	5	sampled, 0.7 / 0.9
SFT	Qwen3-4B	retrieved top-20	SFT	–	–	greedy

Table 14: Bootstrap confidence intervals over questions for the three-run mean of the RITS policy.

Slice	Answer Acc.	Answer 95% CI	Trace score	Trace 95% CI
ID overall	83.1	[80.1, 86.0]	98.7	[98.2, 99.2]
OOD overall	69.5	[66.3, 72.8]	94.0	[93.2, 94.6]
OOD unseen template	60.8	[55.6, 66.3]	93.8	[92.8, 94.7]
OOD unseen tool	78.1	[73.9, 82.1]	94.1	[93.0, 95.2]

G.3 Variability across Independent Training Runs

The RITS rows of Table 1 average three independent training runs of the full recipe. Each run is scored on the test splits under the identical protocol: retrieved top-20 candidate sets, greedy decoding, and the shared five-turn budget. Table 15 lists the per-run results. The three runs land in a narrow band, 82.8 to 83.4 on ID and 68.4 to 70.2 on OOD, with overlapping 95% confidence intervals. Trace scores, execution failures, and call counts vary just as little. The conclusions of Table 1 hold for each run individually. Every run answers more questions correctly than every baseline on ID, trails only Claude Opus 4.8 on OOD, and keeps the highest ID trace score of all systems. The single-policy analyses of this appendix use Run 3.

The runs also agree on which questions they solve. Pairwise agreement on per-question correctness is 88.2 to 90.0% on ID and 81.4 to 89.4% on OOD. All three runs solve 74.6% of ID and 57.4% of OOD questions, and at least one run solves 90.6% and 81.2%, so the union exceeds the best single run by 7.2 and 11.0 points. Majority voting does not realize this headroom, staying within 0.6 points of the best single run at 84.0 on ID and 69.8 on OOD, because errors are correlated across runs.

The failure structure of RQ3 and RQ4 is likewise run-independent. Unseen templates are harder than unseen tools for every run, at 58.8 to 62.0 against 78.0 to 78.4 accuracy, and the OOD no-final-answer cases of Table 24 are run-specific rather than structural: each run leaves 31 to 46 OOD questions unanswered, yet only six questions are unanswered by all three, all instances of the four-hop template that dominates the questions no run solves. The questions unsolved by every run are therefore few and concentrated in the deepest workflow structures.

Table 15: Per-run results of the three independent RITS training runs on the test splits, with 95% bootstrap confidence intervals over questions. Run 3 is used for the single-policy analyses.

	ID			OOD		
Run	Acc. [95% CI]	Trace	OV-F1	Acc. [95% CI]	Trace	OV-F1
Run 1	83.4 [80.2, 86.6]	98.2	97.8	68.4 [64.4, 72.4]	94.7	91.4
Run 2	83.0 [79.8, 86.2]	99.1	99.1	69.8 [65.8, 73.4]	93.8	86.9
Run 3	82.8 [79.6, 86.0]	98.9	98.1	70.2 [66.2, 74.0]	93.4	89.7
Mean	83.1 [80.1, 86.0]	98.7	98.3	69.5 [66.3, 72.8]	94.0	89.3

G.4 Difficulty by Structure and Length

Table 16 disaggregates accuracy by template category. Categories with similar hop counts behave very differently. On ID, Chain-4 templates score 53.0 accuracy while conditional-branch templates score 82.9, and on OOD the unseen-template slice of Chain-4 falls to 11.4 while conditional branches transfer at 100.0. Depth is likewise non-monotonic within a category: Chain-5 templates score 80.7 on ID against 53.0 for Chain-4. Workflow structure therefore explains difficulty better than hop count alone, consistent with the per-item spread in Figure 7. The Chain-4 rows also show the divergence of RQ4 in extreme form. With 53.0 accuracy against a 99.7 trace score in distribution, even familiar four-hop structures are executed correctly and fail at the final answer. Table 17 disaggregates accuracy by trace length. Answer accuracy is not monotonic in trace length, and the unseen-template subset remains the harder one in most length bins, so sequence length does not explain the difficulty either.

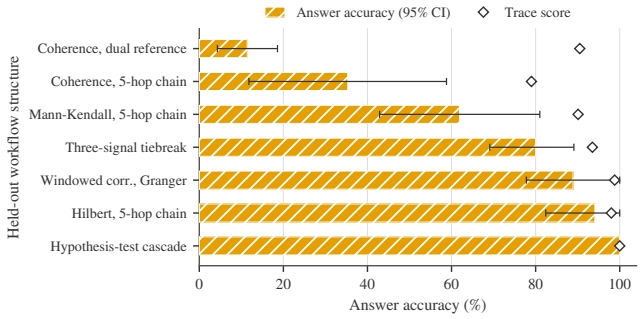


Figure 7: Answer accuracy for representative held-out OOD items under retrieved candidate sets, with 95% confidence intervals. Diamonds mark trace scores. Difficulty varies widely across unseen workflow structures while trace scores stay high.

Table 16: Template-category accuracy for selected template categories.

Split	Category	n	Answer Acc.	95% CI	Trace score
ID	Chain-4	83	53.0	[42.2, 63.9]	99.7
ID	Chain-5	83	80.7	[72.3, 89.2]	99.5
ID	Conditional-branch	105	82.9	[75.2, 89.5]	99.5
ID	Pair-column 2-window cmp	43	97.7	[93.0, 100.0]	100.0
ID	Threshold-relative chain	25	84.0	[68.0, 96.0]	95.6
OOD	Chain-4 / unseen template	70	11.4	[4.3, 18.6]	90.5
OOD	Chain-4 / unseen tool	42	85.7	[73.8, 95.2]	89.9
OOD	Chain-5 / unseen template	38	50.0	[34.2, 65.8]	85.1
OOD	Chain-5 / unseen tool	31	80.6	[67.7, 93.5]	97.2
OOD	Conditional-branch / unseen template	44	100.0	[100.0, 100.0]	100.0
OOD	Multi-probe consensus / unseen tool	42	42.9	[28.6, 57.1]	100.0

Table 17: Length scaling for the full-reward policy. Values are percentages except counts.

Length bin	ID n	ID Acc.	OOD-temp. n	OOD-temp. Acc.	OOD-tool n	OOD-tool Acc.
1–512	24	100.0	12	100.0	17	100.0
513–1024	55	83.6	41	53.7	16	87.5
1025–2048	63	82.5	32	71.9	36	86.1
2049–4096	65	95.4	28	64.3	27	74.1
4097–8192	143	79.7	63	55.6	58	75.9
8193–16384	89	75.3	45	60.0	53	69.8
>16384	61	82.0	29	62.1	43	76.7

G.5 Tool Categories and Domains

Table 18 breaks OOD quality down by tool category. Cross-spectral and cross-correlation tools are the clearest weakness, with trace F1 of 17.8 and 64.3 against 80 to 98 elsewhere. The coherence-style held-out items that collapse in Figure 7 draw on exactly these categories, so the structural failure and the tool-category failure point to the same gap. Both weak categories relate two signals, so the residual tool-level difficulty concentrates in cross-signal analysis. Across data domains, answer accuracy varies from 53.6 to 93.5 with trace scores of 88.5 to 98.0, a narrower spread than the 16.1 to 94.1 across tool categories (Table 19). The residual error therefore tracks the required capability more closely than the data domain.

Table 18: Selected OOD tool-category quality. Ref. counts the reference-trace rows per category. Values except Ref. are percentages.

Tool category	Ref.	Pred. usage	Trace F1	Param. acc.	Answer Acc.
Descriptive Statistics	292	55.6	91.3	91.5	65.1
Distribution	60	11.8	80.0	80.0	83.3
Trend	65	13.2	96.4	96.9	87.7
Smoothing	82	16.8	94.7	92.7	54.9
Seasonality	136	27.2	95.2	98.2	77.2
Spectral	18	3.6	87.0	80.6	38.9
Hilbert	17	3.4	98.0	97.1	94.1
Cross-Correlation	97	18.4	64.3	76.3	33.0
Cross-Spectral	87	10.8	17.8	23.0	16.1
Entropy/Complexity	59	11.2	89.3	89.8	89.8

Table 19: OOD domain breakdown for domains with at least 20 examples. Values are percentages except counts.

Domain	n	Answer Acc.	Trace score
stock	57	71.9	93.7
exchange_rate	37	59.5	93.2
electricity	35	82.9	95.8
ETTh2	32	71.9	92.9
weather	31	93.5	98.0
ETTh1	29	72.4	91.0
PEMS08_flow	28	64.3	88.5
PEMS08_speed	28	53.6	94.7
DailyDelhiClimate	23	82.6	95.9
illness	23	65.2	91.7
PEMS08_occupancy	22	68.2	93.2

G.6 External Benchmark Transfer

External transfer tests whether trace-aware training produces tool-use behavior that generalizes beyond the RITS distribution. Table 20 evaluates the RITS policy zero-shot on the multiple-choice tasks of TimeOmni-1 (Guan et al., 2026) and the weather and finance QA subsets of MTBench (Chen et al., 2025). The comparison system is the pre-RL Qwen3-4B initialization of the policy under the identical protocol, so the gap measures what RL on RITS added. Answers are graded as exact letter match with unanswered questions counted wrong. Trace metrics are not reported because no reference traces exist outside RITS. Benchmarks whose answers are not multiple-choice letters require per-benchmark graders and matched base runs and are not included.

The policy exceeds its base on every task except scenario understanding, by 11.8 to 19.4 points. The mechanism is largely robustness. The policy returns a valid answer on 96 to 100% of questions using 2.7 to 4.2 tool calls per question, while the base leaves 28 to 40% of questions unanswered, and nearly all of decision making, while calling tools up to three times as often on those it attempts. Conditioned on answering, the base can look stronger, 80.7 against 51.5 on the scenario task, but that comparison keeps only the questions the base found easy enough to answer. Transfer strength

Table 20: External-benchmark detail: multiple-choice accuracy of the RITS policy and its pre-RL base, with per-task majority-class and random baselines. Answered: fraction of questions with a committed final answer. The TimeOmni-1 subtotal covers its three multiple-choice tasks.

		Accuracy			Answered			
Benchmark / task	n	Base	RITS	Δ	Base	RITS	Majority	Random
TimeOmni-1, MC subtotal	3,160	34.0	43.4	+9.4	–	–	24.4	∼27
causality discovery	1,600	29.3	45.6	+16.3	72	98	–	33.3
scenario understanding	1,099	55.1	50.4	−4.7	68	98	–	25.0
decision making	461	0.0	19.1	+19.1	0	98	–	25.0
MTBench weather, short	642	31.0	45.2	+14.2	62	100	42.7	25.0
MTBench weather, long	721	32.9	44.7	+11.8	60	99	48.0	25.0
MTBench finance, short	460	63.0	81.1	+18.1	72	98	26.7	25.0
MTBench finance, long	485	56.9	76.3	+19.4	66	96	26.4	25.0

is task-dependent. Finance QA transfers strongly, 50 points above its majority class. Weather QA improves over the base by 12 to 14 points yet stays near its majority class, so we claim only the improvement. Decision making is the weakest transfer, below random even though the base fails it entirely. On scenario understanding the base is higher. The task asks which candidate scenario matches the stated series length. The policy nevertheless computes trend and shape statistics on every question and picks the scenario that best fits them, a pattern behind 92% of its errors. The base, which calls tools on only 64% of questions, more often reasons about the length directly.

The closed-reference column of Table 4 carries the accuracies the benchmark authors report under their own in-context protocols: GPT-4.1 on the TimeOmni-1 ID split and GPT-4o on MTBench. To match this protocol, the TimeOmni-1 rows of Table 4 are restricted to the ID split, whereas Table 20 aggregates ID and OOD. Per task the policy drops only a few points from ID to OOD. The MTBench rows of Table 4 combine the short and long subsets of Table 20 by size-weighted average.

G.7 Trace–Answer Relationship

Table 21 counts the responses on which the trace score and answer correctness disagree. The disagreement runs almost exclusively in one direction: 81 ID and 88 OOD responses score high on the trace yet miss the final answer, against 2 and 4 in the reverse direction. Table 22 reports the per-example correlations between the two metrics. The correlations are accordingly moderate, with Pearson 0.275 and Spearman 0.473 on OOD, and weakest in the unseen-tool slice, where the trace score is least predictive of answer success. The audit also isolates a third bucket of syntactic execution failures, rare on ID (1 response) but visible on OOD (51), in line with the higher failure rate under retrieved candidate sets. Under answer-only evaluation these 169 high-trace responses would be indistinguishable from tool-use failures. The trace metrics show that the procedure succeeded and the final-answer step failed. The two metrics measure different capabilities and are therefore reported together throughout the paper.

Table 21: Trace metric audit. High-trace denotes a trace score of at least 0.9, and low-trace below 0.5.

Audit bucket	ID count	OOD count
High-trace wrong-answer	81	88
Low-trace correct-answer	2	4
Syntactic execution failure	1	51

Table 22: Trace-answer correlation on OOD predictions.

Slice	Answer Acc.	Trace score	Pearson	Spearman
Overall OOD	70.2	93.4	0.275	0.473
Unseen template	62.0	93.5	0.421	0.666
Unseen tool	78.4	93.2	0.183	0.194

G.8 Reward-Independent Trace Metrics

The trace score reported throughout the paper is the trace term r_τ of the training reward. This subsection rescores the same predictions with two metrics that share no quantity with that reward: neither uses partial credit, argument similarity, output comparison, or a length penalty. *Trace-EM* scores a response 1 only when every reference call is reproduced with the exact tool name and arguments, the dependency order is respected, and no extra call is issued, following the exact-match convention of function-calling benchmarks (Patil et al., 2025); any deviation scores 0. *Ordered-valid tool F1 (OV-F1)* augments the tool-name micro-F1 of tool-use benchmarks (Wu et al., 2024; Basu et al., 2025) with two behavioral gates: a predicted call can match a reference call only when it executes without error and appears in an order consistent with the reference dependencies, where calls within a hop may run in any order. Formally, OV-F1 reuses the notation and the matching procedure of Appendix F: the reference nodes V in dependency levels $L_1, \ldots, L_m$ are matched to the predicted calls $c_1, \ldots, c_T$ by Algorithm 1, with the graded pair score of Equation 10 replaced by the binary rule

$$s^{\mathrm{OV}}(v, c) \;=\; \mathbb{1}\big[\mathrm{name}(c) = \ell(v)\big] \cdot \mathbb{1}\big[\mathrm{ok}(c)\big] \;\in\; \{0, 1\}, \tag{11}$$

where $\mathrm{ok}(c)$ holds when c executes without error. The level-wise assignment leaves calls within a hop free to run in any order, while the cursor bars calls issued before their dependencies. With M_q the matched pairs of question q, and T_q and V_q its predicted calls and reference nodes, the score aggregates as corpus-level micro-F1 over the split,

$$P = \frac{\sum_q |M_q|}{\sum_q T_q}, \qquad R = \frac{\sum_q |M_q|}{\sum_q |V_q|}, \qquad \text{OV-F1} = \frac{2PR}{P + R}, \tag{12}$$

so failed, out-of-order, and extra calls lose the ability to match but remain in the denominators. A failed call that a later retry recovers therefore costs precision only; a failure left unrecovered costs recall as well. The execution gate follows the executable-accuracy convention of function-calling evaluation (Patil et al., 2025). The composition of the two gates and the denominator convention are specific to RITS. The only quantities shared with the training reward are the score-free alignment procedure and the dependency levels, which the dataset generator fixes independently of any reward.

Table 23 reports both metrics for the systems of Table 1. On ID, RITS leads every system by a wide margin on both metrics. On OOD, RITS attains the best OV-F1, with Claude Opus 4.8 statistically tied at 0.2 points below, and every other system further behind. OOD Trace-EM instead ranks SFT first, at 70.8 against 63.1 for RITS. The reversal comes from the no-extra-call condition rather than from execution quality. A single extra call sets Trace-EM to 0, which favors the verbatim reproduction that SFT is trained for, whereas the extra calls of RITS are functional: RITS answers 79 of its zero-Trace-EM questions correctly against 34 for SFT. Removing the order constraint changes no system on either split, so call order contributes nothing to these rankings.

Three analyses support the independence of these rankings from the training reward. The ablation variant trained without the trace term (R3 of Table 2) reaches an OOD OV-F1 of 97.6, close to the 99.1 of the trace-trained variant, while the answer-only variant R1 stays at 66.2. Claude Opus 4.8, which never encounters the reward, comes within 0.2 points of RITS on OOD OV-F1. Rescoring without the order gate leaves the RITS score unchanged, so the alignment rule does not favor the trained policy.

Table 23: Reward-independent trace metrics for the systems of Table 1. Trace-EM is the fraction of responses reproducing the reference trace exactly with no extra call; OV-F1 is micro-F1 over tool names counting only calls that execute without error in a dependency-consistent order. Values are percentages.

	Trace-EM		OV-F1	
Method	ID	OOD	ID	OOD
GPT-5.6-sol	48.0	58.8	87.0	87.8
Gemini 3.1 Pro	44.0	54.4	89.9	88.0
Claude Opus 4.8	50.4	61.2	90.1	89.1
Qwen3-4B	20.6	12.8	18.6	33.2
Ministral-3-8B-Reasoning	43.6	45.2	75.9	79.9
Qwen3.6-27B	39.6	47.8	83.0	88.0
SFT	56.4	**70.8**	81.3	87.8
RITS (ours)	**92.0**	63.1	**98.3**	**89.3**

G.9 Failure Mode Analysis

Failed responses are categorized into format collapse, missing data-loading call, invalid arguments, parameter mis-binding, wrong tool with right answer, incorrect hop ordering, candidate-set miss, and tool-execution failure. Table 24 aggregates the dominant buckets. Answer and normalization errors are the largest class on both splits, at 15.0% on ID and 12.6% on OOD. The bucket specific to OOD, at 9.2%, is the case where no final answer is given. The policy completes its tool calls but does not commit to an answer on unfamiliar structures. Selection-side failures stay small, with name misselection at 3.0%, omitted calls at 2.4%, and candidate-set misses at 1.0% on OOD. The distribution matches the final-answer bottleneck of RQ4. Errors arise after the tool calls more often than in choosing or executing them.

Table 24: Failure-mode breakdown for the RITS policy. Dashes mark categories that are not major buckets for that split.

Failure mode	ID (%)	OOD (%)
Correct	83.0	70.2
Answer / normalization error	15.0	12.6
No final answer	–	9.2
Name misselection	0.4	3.0
Omitted calls	1.0	2.4
Candidate-set miss	–	1.0
Runtime exception	0.4	–

G.10 Item-Level Contrast between SFT and RITS

We compare the SFT policy of Table 1 with RITS question by question under the shared protocol: retrieved top-20 candidate sets, five turns, greedy decoding, and both policies served in their training-time format. Table 25 shows the question-level win–loss counts. RITS answers 171 of the 231 ID questions that SFT misses, while SFT answers 26 that RITS misses; the asymmetry is significant on both splits (exact McNemar $p < 10^{-26}$ on ID and $p = 0.005$ on OOD). On the questions that only RITS solves, both policies spend a median of six tool calls, so the gap reflects accuracy rather than budget.

Three behavioral differences drive the gap. First, error recovery: among ID questions in which the policy encounters a tool error, RITS reaches a final answer in 13 of 14 cases and SFT in 19 of 129. Reference traces contain no errors, so imitation never observes the error state. Second, precision on deep hops: 85% of the RITS-only ID wins are four-hop questions, and on the 76 of them where SFT completes its calls but answers wrong, the executed calls match the reference nearly as closely as on its correct answers. The workflow looks right, but a parameter binding or an intermediate reading goes wrong. Third, termination: SFT leaves 134 ID questions unanswered against 2 for RITS.

Table 25: Question-level win–loss counts between SFT and RITS under the shared evaluation protocol (500 questions per split).

Split	Both correct	RITS only	SFT only	Both wrong
ID	243	171	26	60
OOD	260	91	56	93

H Prompts and Worked Examples

This appendix reproduces the prompts and data artifacts of the RITS pipeline: the policy prompt, a runtime tool schema, and a complete worked example with its reference trace and intermediate answers. Dataset paths are abbreviated to repository-relative form.

Policy prompt. Every tool-using system receives the same two-message prompt, shown in Figure 8 (top): a fixed system message and a user message that states the data context and the question.

Runtime tool schema. Tool schemas are generated from the registry at load time (Appendix B). Figure 8 (bottom) shows the generated output for `adf_test`, one of the tools in the worked example, with unset fields elided. Enum-constrained parameters carry their admissible values, and domain tags are folded into the description.

Worked example: a conditional-branch question. Figure 9 shows a retained three-hop question from the `stationarity_branch` template, the conditional-branch structure of Figure 5, together with its reference trace. Its reference tools are `load_csv_data`, `adf_test`, `kpss_test`, and `hurst`. The candidate set presented to the agent adds distractors up to the cap of Section 3.5.

Hops 2 and 3 are independent and may execute in either order. The branch condition then selects the third analysis from their outputs. Here the ADF p-value of 0.914056 (at least 0.05) and the KPSS p-value of 0.01 (at most 0.05) both indicate non-stationarity. The non-stationary branch is therefore selected, and the Hurst exponent supplies the final answer. The other two branches (Welch dominant period, spectral entropy) appear in the question but not in the reference trace, so an agent that executes an unselected branch loses trace precision, exactly the behavior the matching of Appendix F penalizes.

Baseline prompts. The closed-model function-calling baselines reuse the identical policy prompt. Candidate-set tool schemas are supplied through each provider's native function-calling interface rather than inlined in the system message. Question-only prompting and the code-execution baseline replace the tool protocol as shown in Figure 10. The latter runs two roles, a reasoning agent (the shared Qwen3-4B base) that delegates every computation and a coding assistant (frozen Qwen3-Coder-30B) that executes it. All baselines are scored by the same reward module as the policy, so answer accuracy and trace scores are directly comparable.

Policy prompt (system and user messages)

```
[system]
You are a time-series analyst. To answer the user's question you MUST
use the provided tools. Call `load_csv_data` first to load the
↪  dataset,
then chain the analysis tools needed to resolve each sub-step. Do not
guess numerical values -- compute them via tools. When finished, emit
the final answer inside <answer>...</answer> as a single short token
or number.

[user]
Dataset at: data/exchange_rate/exchange_rate.csv
Domain: exchange_rate
Row window in focus: [1897, 3794)

Question: <question text>
```

Runtime tool schema: `adf_test`

```
{"type": "function",
 "function": {
   "name": "adf_test",
   "description": "Augmented Dickey-Fuller test. `regression` in
                   {c, ct, n}; Tags: univariate, hypothesis,
                   stochastic.",
   "parameters": {
     "type": "object",
     "properties": {
       "column":     {"type": "string",
                      "description": "Column name to analyze."},
       "start":      {"type": "string", "description": "Slice start
       ↪  --
                      integer index or timestamp string. Optional."},
       "end":        {"type": "string", "description": "Slice end --
                      integer index or timestamp string. Optional."},
       "regression": {"type": "string", "enum": ["c", "ct", "n"],
                      "description": "Regression specification
                      (c=constant, ct=constant+trend, n=no
                      ↪  constant)."},
       "output":     {"type": "string", "enum": ["pvalue", "stat"],
                      "description": "Returned statistic
                      (pvalue or stat)."}},
     "required": ["column", "regression", "output"]}}}
```

Figure 8: The two-message policy prompt (top) and the runtime tool schema generated from the registry entry for `adf_test` (bottom). The system message fixes the tool-use protocol and the answer format; at runtime, the candidate set's schemas are embedded in the system message under a `# Tools` section. Enum-constrained schema parameters carry their admissible values, and domain tags are appended to the description.

Question (`stationarity_branch`, three hops, conditional branch)

For the time series at data/exchange_rate/exchange_rate.csv (exchange_rate), on column 'new_zealand' over the Q2 range of rows [1897, 3794), first compute the Augmented Dickey-Fuller test p-value, fit with a constant in the test regression, on column 'new_zealand' in that window and the KPSS stationarity-test p-value, fit with a constant in the test regression, on column 'new_zealand' in that window.

If the ADF p-value is below 0.05 AND the KPSS p-value is above 0.05 (both indicate stationarity), report the Welch-method dominant period of the same column over the same window, rounded to 4 decimal places.

Otherwise, if the ADF p-value is at least 0.05 AND the KPSS p-value is at most 0.05 (both indicate non-stationarity), report the rescaled-range Hurst exponent of the same column over the same window, rounded to 4 decimal places.

Otherwise (the two tests disagree), report the spectral entropy of the same column over the same window, rounded to 4 decimal places.

Reference trace and stored intermediate answers

```
1. load_csv_data(file_path=data/exchange_rate/exchange_rate.csv)
2. adf_test(column=new_zealand, start=1897, end=3794,
            regression=c, output=pvalue)          -> 0.914056
3. kpss_test(column=new_zealand, start=1897, end=3794,
             regression=c, output=pvalue)         -> 0.01
4. hurst(column=new_zealand, start=1897, end=3794,
         estimator=rs)                            -> 0.9954

final answer: 0.9954
```

Figure 9: Worked example from the `stationarity_branch` template (conditional branch, three hops). Top: the question. Bottom: the reference trace, which stores the intermediate answer of every hop and contains no call from the unselected branches.

Question-only prompt

```
[system]
You are answering an analytical question about a time series. You do
↪  NOT
have access to any tools or to the underlying series data; answer
↪  from the
question text alone. Reason briefly if it helps, then give your
↪  single best
final answer inside <answer> and </answer> tags.

[user]
<question text>
```

Code-execution baseline prompts (reasoning agent and coding assistant)

```
[reasoning-agent system]
You are a time-series analyst answering a multi-hop question about a
↪  series
stored in a CSV file. You CANNOT see the data and you CANNOT run code
yourself; you delegate every computation to a Python coding
↪  assistant.

Protocol -- follow it exactly:
- Reason step by step inside <think>...</think>.
- Whenever you need a value computed from the data, emit exactly ONE
  <call_coding>...</call_coding> block that describes the computation
  ↪  in
  plain English. Be specific: state the CSV path, the column, the row
  window [start, end), and the exact quantity to compute, and ask the
  assistant to print the result to stdout.
- You will then receive a '[Coding Result]' message with the printed
  output. Use it to continue.
- You may issue several <call_coding> blocks, one per turn.
- Once you have the final answer, output it inside
↪  <answer>...</answer>
  as a single short token or number, with no other text inside the
  ↪  tags.

[coding-assistant system]
You are a Python coding assistant specialized in time-series data
analysis. Given a task specification from a reasoning agent, write
↪  clean,
correct, and efficient Python code that solves the task. Always wrap
↪  your
code in a single python code block. The code must print the final
↪  result
to stdout. Do not include explanations outside the code block.
```

Figure 10: Baseline prompts. Top: question-only prompting, which removes tools and data access. Bottom: the two roles of the code-execution baseline; the reasoning agent delegates computations in plain English and the coding assistant returns executed results.

Qualitative case studies. The cases below are drawn from the retrieved-condition evaluation pool and selected by what the trace metrics reveal rather than by which system wins. The full candidate pool and transcripts accompany the release. Case 1 shows why the trace is necessary. The policy answers correctly, and every baseline does too, yet the trace score (42.0) records that the deciding median hops were never executed. The values its reasoning cites for them, 48.0 and 36.45, are read off the intermediate maximum arrays. The true medians are 18.41 and 17.68. Answer-level scoring cannot detect this. Case 2 shows the final-answer bottleneck of RQ4 in pure form. A reference-exact seven-call trajectory (trace score 100) computes both operands correctly, then flips the direction of the final comparison. The answer-only ablation R1 on the same question never executes the second branching test at all, failing at the procedure rather than at the final answer. Case 3 contrasts the policy with the answer-only ablation R1 on the same question. The policy binds both branching tests correctly, while R1 runs the follow-up test on the losing column, executes a test the question never asks for, and reports the value of the rejected branch. Answer-level evaluation records this only as an incorrect answer, whereas trace-level evaluation identifies the step at which the workflow deviated.

Case 1: Trace necessity under a correct answer (triple-max tiebreak, four hops, OOD). Reference answer: Q3.

Question

For the time series at data/stock/stock_data.csv (stock), among the Q1 range of rows [0, 2149), the Q3 range of rows [4298, 6447), the H2 range of rows [4298, 8597), take the two ranges that yield the largest maximum of column 'vix'; among those two ranges, return the one that yields the larger median of the same column. Reply with only the window label ('Q1', 'Q3', 'H2').

RITS (ours) — answer Q3 — ✓ correct answer; trace score 42: no median call executed

```
<think>


</think>

<tool_call>
{"name": "load_csv_data", "arguments": {"file_path":
  "data/stock/stock_data.csv", "value_columns": ["vix"]}}
</tool_call>user
<tool_response>
{"success": true, "shape": [8597, 1], "columns": ["vix"],
↪  "missing_values": {"vix": 0},
  "timestamp_detected": true, "date_range": {"start": "1990-01-03
  ↪  00:00:00", "end":
  "2024-02-16 00:00:00"}}
</tool_response>
assistant
<think>
Okay, let's tackle this step by step. First, I need to find the two
↪  ranges among Q1, Q3,
  and H2 that have the largest maximum value for the 'vix' column.
  ↪  Then, between those two
  ranges, I need to determine which one has the larger median for the
  ↪  same column.

The ranges given are:
- Q1: rows 0 to 2149
- Q3: rows 4298 to 6447
- H2: rows 4298 to 8597

I'll start by finding the maximum values for each of these ranges.
↪  Then, I'll identify the
  two largest maxima and check their medians to see which one is
  ↪  larger.
```

```
</think>

<tool_call>
{"name": "resample", "arguments": {"column": "vix", "start": "0",
↪  "end": "2149", "freq":
  "yearly", "how": "max"}}
</tool_call>
<tool_call>
{"name": "resample", "arguments": {"column": "vix", "start": "4298",
↪  "end": "6447",
  "freq": "yearly", "how": "max"}}
</tool_call>
<tool_call>
{"name": "resample", "arguments": {"column": "vix", "start": "4298",
↪  "end": "8597",
  "freq": "yearly", "how": "max"}}
</tool_call>user
<tool_response>
{"value": [36.47, 36.2, 21.02, 17.3, 23.87, 15.74, 21.99, 38.2,
↪  28.69]}
</tool_response>
<tool_response>
{"value": [31.09, 80.86, 56.65, 45.79, 48.0, 26.66, 20.49, 26.25,
↪  22.39]}
</tool_response>
<tool_response>
{"value": [31.09, 80.86, 56.65, 45.79, 48.0, 26.66, 20.49, 26.25,
↪  40.74, 28.14, 16.04,
  37.32, 25.45, 82.69, 37.21, 36.45, 26.52, 15.85]}
</tool_response>
assistant
<think>
Okay, let's break down the problem. First, I need to identify the two
↪  ranges (Q1, Q3, H2)
  that have the largest maximum value for the 'vix' column.

Looking at the resampled maximum values for each range:
- Q1 has a maximum of 36.47.
- Q3 has a maximum of 80.86.
- H2 has a maximum of 82.69.

The two largest maxima are from Q3 and H2. Next, I need to check the
↪  medians of these two
  ranges.

For Q3, the median of the 'vix' column in the range [4298, 6447) is
↪  48.0. For H2, the
  median in the range [4298, 8597) is 36.45.

Comparing these medians, Q3 has a higher median. Therefore, the
↪  answer is Q3.
</think>

<answer>Q3</answer>
```

Case 2: Finalization flip after a perfect trace (Hilbert quintuple chain, five hops, OOD). Reference answer: lower.

Question

For the time series at data/big_company_stock/Microsoft/Microsoft.csv (big_company_stock.Microsoft), is the STL seasonal strength ratio using a 24-step seasonal cycle – computed, within whichever of the Q1 range of rows [0, 1459) and the Q2 range of rows [1459, 2919) yields the larger Hilbert envelope mean, of whichever of columns 'High' and 'Low' has the larger Hilbert envelope standard deviation over the Q1 range of rows [0, 1459) – strictly higher or lower than the STL seasonal strength ratio using a 24-step seasonal cycle of that same column over the Q2 range of rows [1459, 2919)? Reply with exactly 'higher' or 'lower'.

RITS (ours) — answer higher — ✗ incorrect; trace score 100: all seven calls reference-exact

```
<think>


</think>

<tool_call>
{"name": "load_csv_data", "arguments": {"file_path":
  "data/big_company_stock/Microsoft/Microsoft.csv",
  "value_columns": ["High", "Low"]}}
</tool_call>user
<tool_response>
{"success": true, "shape": [5839, 2], "columns": ["High", "Low"],
↪  "missing_values":
  {"High": 0, "Low": 0}, "timestamp_detected": true, "date_range":
  ↪  {"start": "1970-01-01
  00:00:00.000000015", "end": "1970-01-01 00:00:00.000000344"}}
</tool_response>
assistant
<think>
Okay, let's tackle this step by step. First, I need to determine
↪  which of the 'High' or
  'Low' columns has a larger Hilbert envelope standard deviation over
  ↪  the Q1 range of rows
  from 0 to 1459. I'll use the hilbert_envelope_std function for each
  ↪  column over that
  range.

Next, after identifying the column with the larger standard
↪  deviation, I'll check which of
  the two ranges (Q1 or Q2) yields a larger Hilbert envelope mean.
  ↪  Then, I'll compute the
  STL seasonal strength ratio using a 24-step seasonal cycle for the
  ↪  selected column over
  both ranges. Finally, I'll compare the two ratios to determine if
  ↪  the STL seasonal
  strength ratio from the range with the larger Hilbert envelope mean
  ↪  is strictly higher
  or lower than the one from the other range.
</think>

<tool_call>
{"name": "hilbert_envelope_std", "arguments": {"column": "High",
↪  "start": "0", "end":
  "1459"}}
</tool_call>
<tool_call>
```

```
{"name": "hilbert_envelope_std", "arguments": {"column": "Low",
↪  "start": "0", "end":
  "1459"}}
</tool_call>user
<tool_response>
{"value": 6.5792180819789765}
</tool_response>
<tool_response>
{"value": 6.190137626750207}
</tool_response>
assistant
<think>
Okay, let's break down the problem step by step. First, I need to
↪  determine which of the
  'High' or 'Low' columns has the larger Hilbert envelope standard
  ↪  deviation over the Q1
  range (rows 0 to 1459). From the previous results, the Hilbert
  ↪  envelope standard
  deviation for the 'High' column is approximately 6.5792, and for
  ↪  the 'Low' column, it's
  approximately 6.1901. Since 'High' has a larger standard deviation,
  ↪  I'll focus on that
  column.

Next, I need to check which of the Q1 or Q2 ranges (rows 0 to 1459 or
↪  1459 to 2919) yields
  a larger Hilbert envelope mean. I'll use the hilbert_envelope_mean
  ↪  function on both
  ranges for the 'High' column.

After that, I'll compute the STL seasonal strength ratio using a
↪  24-step seasonal cycle
  for the 'High' column over both the Q1 and Q2 ranges. Finally, I'll
  ↪  compare the two
  ratios to determine if the STL seasonal strength ratio from the
  ↪  range with the larger
  Hilbert envelope mean is strictly higher or lower than the other.
</think>

<tool_call>
{"name": "hilbert_envelope_mean", "arguments": {"column": "High",
↪  "start": "0", "end":
  "1459"}}
</tool_call>
<tool_call>
{"name": "hilbert_envelope_mean", "arguments": {"column": "High",
↪  "start": "1459", "end":
  "2919"}}
</tool_call>user
<tool_response>
{"value": 30.36020649500698}
</tool_response>
<tool_response>
{"value": 27.177217395198465}
</tool_response>
assistant
<think>
Okay, let's see. First, I need to determine which of the two ranges,
↪  Q1 (rows 0 to 1459)
  or Q2 (rows 1459 to 2919), has a larger Hilbert envelope mean for
  ↪  the 'High' column. The
  results showed that the Hilbert envelope mean for Q1 is
  ↪  approximately 30.3602, and for
```

```
  Q2, it's about 27.1772. Since Q1 has a larger mean, I'll focus on
  ↪ the STL seasonal
  strength ratio for the 'High' column using a 24-step seasonal cycle
  ↪ on both ranges.

Next, I need to calculate the STL seasonal strength ratio for the
↪ 'High' column. I'll use
  the stl_seasonal_strength function for both Q1 and Q2 ranges with a
  ↪ period of 24. Then,
  I'll compare the two ratios to determine if the one from Q1 is
  ↪ strictly higher or lower
  than the one from Q2.
</think>

<tool_call>
{"name": "stl_seasonal_strength", "arguments": {"column": "High",
↪ "start": "0", "end":
  "1459", "period": 24}}
</tool_call>
<tool_call>
{"name": "stl_seasonal_strength", "arguments": {"column": "High",
↪ "start": "1459", "end":
  "2919", "period": 24}}
</tool_call>user
<tool_response>
{"value": 0.0}
</tool_response>
<tool_response>
{"value": 0.08892793552025646}
</tool_response>
assistant
<think>
Okay, let's break this down. First, I need to compare the STL
↪ seasonal strength ratios
  from the two ranges. The Q1 range (rows 0 to 1459) has a STL
  ↪ seasonal strength of 0.0,
  and the Q2 range (rows 1459 to 2919) has a ratio of approximately
  ↪ 0.0889.

Since 0.0889 is strictly higher than 0.0, the answer is "higher".
</think>

<answer>higher</answer>
```

R1 (answer-only ablation, same base) — answer higher — ✗ incorrect; trace score 50: the envelope-mean comparison is never executed

```
<think>
Okay, this is a complex question. Let me break it down step by step.
↪ The user wants to
  compare the STL seasonal strength ratio using a 24-step cycle in
  ↪ two different ranges
  (Q1 and Q2) of the Microsoft stock data. The key is to determine
  ↪ which column ('High' or
  'Low') and which range (Q1 or Q2) gives a higher Hilbert envelope
  ↪ mean, then compare the
  STL seasonal strength ratios.

First, I need to find which column (High or Low) has a larger Hilbert
↪ envelope standard
  deviation in Q1. Then, based on that, check which range (Q1 or Q2)
  ↪ gives a larger
```

```
  Hilbert envelope mean. Once that's determined, compute the STL
  ↪ seasonal strength ratio
  for that column and cycle in both Q1 and Q2 ranges. Finally,
  ↪ compare these two ratios to
  see if the first is higher or lower than the second.

I need to use the hilbert_envelope_std function for Q1 to find which
↪ column has higher
  standard deviation. Then, use the hilbert_envelope_std function
  ↪ again for Q2 to find the
  mean. Then, use the stl_seasonal_strength function for both ranges
  ↪ to compare the
  ratios.
</think>

<tool_call>
{"name": "hilbert_envelope_std", "arguments": {"column": "High",
↪ "start": "0", "end":
  "1459"}}
</tool_call>
<tool_call>
{"name": "hilbert_envelope_std", "arguments": {"column": "Low",
↪ "start": "0", "end":
  "1459"}}
</tool_call>
<tool_call>
{"name": "stl_seasonal_strength", "arguments": {"column": "High",
↪ "period": 24, "start":
  "0", "end": "1459"}}
</tool_call>
<tool_call>
{"name": "stl_seasonal_strength", "arguments": {"column": "High",
↪ "period": 24, "start":
  "1459", "end": "2919"}}
</tool_call>
<answer>higher</answer>user
<tool_response>
{"error": "No data loaded. Call load_csv_data first."}
</tool_response>
<tool_response>
{"error": "No data loaded. Call load_csv_data first."}
</tool_response>
<tool_response>
{"error": "No data loaded. Call load_csv_data first."}
</tool_response>
<tool_response>
{"error": "No data loaded. Call load_csv_data first."}
</tool_response>
assistant
<think>
Okay, the user is asking about a time series analysis on Microsoft
↪ stock data. The
  previous attempt didn't work because the data wasn't loaded. I need
  ↪ to first load the
  dataset using load_csv_data. Then, proceed with the required steps.

First, I need to load the data from the given file path. Once the
↪ data is loaded, I can
  use the hilbert_envelope_std function on both 'High' and 'Low'
  ↪ columns for the Q1 range.
  Then, determine which column has a higher Hilbert envelope standard
  ↪ deviation. Next,
  check which of the Q1 or Q2 ranges (based on that column) has a
  ↪ larger Hilbert envelope
```

```
  mean. Then, compute the STL seasonal strength ratio for that column
   ↪  and cycle in both Q1
  and Q2 ranges. Finally, compare the two ratios to answer if it's
   ↪  higher or lower.
</think>

<tool_call>
{"name": "load_csv_data", "arguments": {"file_path":
  "data/big_company_stock/Microsoft/Microsoft.csv"}}
</tool_call>user
<tool_response>
{"success": true, "shape": [5839, 5], "columns": ["High", "Low",
↪  "Close", "Adj Close",
  "Volume"], "missing_values": {"High": 0, "Low": 0, "Close": 0, "Adj
   ↪  Close": 0, "Volume":
  0}, "timestamp_detected": true, "date_range": {"start": "1970-01-01
   ↪  00:00:00.000000015",
  "end": "1970-01-01 00:00:00.000000344"}}
</tool_response>
assistant
<think>
Okay, the data was successfully loaded. Now I need to proceed with
↪  the analysis. The next
  step is to calculate the Hilbert envelope standard deviation for
   ↪  both 'High' and 'Low'
  columns in the Q1 range [0, 1459). Then, determine which column has
   ↪  a higher value.
  After that, check which of the Q1 or Q2 ranges (based on that
   ↪  column) has a larger
  Hilbert envelope mean. Then compute the STL seasonal strength ratio
   ↪  for that column and
  cycle in both Q1 and Q2 ranges. Finally, compare the two ratios to
   ↪  see if it's higher or
  lower.
</think>

<tool_call>
{"name": "hilbert_envelope_std", "arguments": {"column": "High",
↪  "start": "0", "end":
  "1459"}}
</tool_call>
<tool_call>
{"name": "hilbert_envelope_std", "arguments": {"column": "Low",
↪  "start": "0", "end":
  "1459"}}
</tool_call>
<tool_call>
{"name": "stl_seasonal_strength", "arguments": {"column": "High",
↪  "period": 24, "start":
  "0", "end": "1459"}}
</tool_call>
<tool_call>
{"name": "stl_seasonal_strength", "arguments": {"column": "High",
↪  "period": 24, "start":
  "1459", "end": "2919"}}
</tool_call>user
<tool_response>
{"value": 6.5792180819789765}
</tool_response>
<tool_response>
{"value": 6.190137626750207}
</tool_response>
<tool_response>
{"value": 0.0}
```

```
</tool_response>
<tool_response>
{"value": 0.08892793552025646}
</tool_response>
assistant
<think>
Okay, let's break this down. The first two tool calls were for
↪  Hilbert envelope standard
  deviation on 'High' and 'Low' in Q1. 'High' had a higher value
  ↪  (6.58 vs 6.19). So we
  pick 'High' as the column. Next, the STL seasonal strength for Q1
  ↪  (0-1459) was 0.0, and
  for Q2 (1459-2919) it was 0.0889. Since 0.0889 is higher than 0.0,
  ↪  the answer is
  'higher'.
</think>

<answer>higher</answer>
```

Case 3: DAG-failure localization against the answer-only ablation (hypothesis-test consensus branch, four hops, ID). Reference answer: 0.003.

Question

For the time series at data/big_company_stock/Amazon/Amazon.csv (big_company_stock.Amazon), what is the last non-NaN value of the 30-step trailing rolling-window Bessel-corrected sample variance – computed, within whichever of rows [0, 1459) and rows [0, 2919) yields the larger Engle ARCH-LM test p-value using 5 lagged squared residuals, of whichever of columns 'High' and 'Low' has the larger Wald-Wolfowitz runs-test p-value over rows [0, 1459), rounded to 3 decimals? Reply with the value rounded to 3 decimals (e.g. '1.234' or '-0.057').

RITS (ours) — answer 0.003 — ✓ correct; trace score 100: six reference-exact calls, both branching tests bound correctly

```
<think>


</think>

<tool_call>
{"name": "load_csv_data", "arguments": {"file_path":
  "data/big_company_stock/Amazon/Amazon.csv",
  "value_columns": ["High", "Low"]}}
</tool_call>user
<tool_response>
{"success": true, "shape": [5839, 2], "columns": ["High", "Low"],
↪  "missing_values":
  {"High": 0, "Low": 0}, "timestamp_detected": true, "date_range":
  ↪  {"start": "1970-01-01
  00:00:00", "end": "1970-01-01 00:00:00.000000187"}}
</tool_response>
assistant
<think>
Okay, let's tackle this step by step. First, I need to determine
↪  which of the 'High' or
  'Low' columns has a larger Wald-Wolfowitz runs-test p-value over
  ↪  the rows from 0 to
  1459. I'll use the runs_test_pvalue function for each column with
  ↪  the specified range.
```

```
Next, after identifying the column with the larger p-value, I'll
↪  check the Engle ARCH-LM
  test p-value using 5 lagged squared residuals for both segments:
  ↪  rows 0 to 1459 and rows
  0 to 2919. I'll then select the segment with the larger p-value and
  ↪  compute the 30-step
  trailing rolling-window Bessel-corrected sample variance. Finally,
  ↪  I'll find the last
  non-NaN value from this variance and round it to three decimal
  ↪  places.
</think>

<tool_call>
{"name": "runs_test_pvalue", "arguments": {"column": "High", "start":
↪  "0", "end": "1459"}}
</tool_call>
<tool_call>
{"name": "runs_test_pvalue", "arguments": {"column": "Low", "start":
↪  "0", "end": "1459"}}
</tool_call>user
<tool_response>
{"value": 1.3527559046617671e-300}
</tool_response>
<tool_response>
{"value": 6.956040358998316e-294}
</tool_response>
assistant
<think>
Okay, let's break down the problem step by step. First, I need to
↪  determine which of the
  'High' or 'Low' columns has the larger Wald-Wolfowitz runs-test
  ↪  p-value over the rows
  from 0 to 1459. From the previous results, the p-value for the
  ↪  'High' column is
  1.3527559046617671e-300 and for the 'Low' column it's
  ↪  6.956040358998316e-294. Since
  6.956040358998316e-294 is larger, I'll focus on the 'Low' column.

Next, I need to check the Engle ARCH-LM test p-value using 5 lagged
↪  squared residuals for
  both segments: rows 0 to 1459 and rows 0 to 2919. I'll use the
  ↪  arch_lm_test_pvalue
  function for each segment and then determine which one has the
  ↪  larger p-value. After
  that, I'll compute the 30-step trailing rolling-window
  ↪  Bessel-corrected sample variance
  for the selected segment and find the last non-NaN value, rounding
  ↪  it to three decimal
  places.
</think>

<tool_call>
{"name": "arch_lm_test_pvalue", "arguments": {"column": "Low",
↪  "start": "0", "end":
  "1459", "lag": 5}}
</tool_call>
<tool_call>
{"name": "arch_lm_test_pvalue", "arguments": {"column": "Low",
↪  "start": "0", "end":
  "2919", "lag": 5}}
</tool_call>user
<tool_response>
{"value": 6.262231687836997e-308}
</tool_response>
```

```
<tool_response>
{"value": 0.0}
</tool_response>
assistant
<think>
Okay, let's see. First, I need to determine which of the two segments (rows 0 to 1459 or 0
  to 2919) has the larger Engle ARCH-LM test p-value for the 'Low' column with 5 lagged
  squared residuals. The p-value for the first segment (0 to 1459) is
  6.2622316878369 97e-308, and for the second segment (0 to 2919) it's 0.0. Since
  6.262231687836997e-308 is larger, I'll work with that segment.

Next, I need to compute the 30-step trailing rolling-window Bessel-corrected sample
  variance for the 'Low' column over the rows 0 to 1459. The Bessel-corrected sample
  variance uses the 'var' operation with a window of 30. I'll apply this and then find the
  last non-NaN value from the results. Finally, I'll round this value to three decimal
  places.
</think>

<tool_call>
{"name": "rolling_window", "arguments": {"column": "Low", "start": "0", "end": "1459",
  "op": "var", "window": 30}}
</tool_call>user
<tool_response>
{"value": [0.1087522191641714, 0.09907908131550897, 0.08308389306094553,
  0.08308389306094553, 0.08316414141364278, 0.08477069863215057, 0.08513504661423174,
  0.08489545202521208, 0.08357953466792352, 0.08288799425035177, 0.08200016446566248,
  0.08070760129529635, 0.08143247604473323, 0.08295944879898148, 0.07988858060693357,
  0.0825611170899467, 0.08653133071985147, 0.08826060369508446, 0.08809677734911521,
  0.08504370464907397, 0.07940078322487078, 0.08119845714103704, 0.08592041771242816,
  0.07732436442281475,...(truncated)...894919028610356, 0.002388282270266123,
  0.002500286300825084, 0.0022772565893646917, 0.002073803603130199,
  0.0019120013774341475, 0.001668297262943853, 0.0014575237532211986,
  0.0017700847997400636, 0.0023394343332882976, 0.002668378351441952,
  0.00278630202005124, 0.0028296161668733733, 0.002858231039750938, 0.002920589250137075,
  0.002893278865320913, 0.0028502445435687114, 0.0028684846840188977,
  0.0027984817201022055, 0.0025956406647055365, 0.0026098309081175374, 0.002742798014454043,
  0.0029898366717301225]}
</tool_response>
assistant
<think>
Okay, let's break down the problem. First, I need to find the last non-NaN value from the
  rolling-window variance results provided. The data shows a series of numbers, and the
  last few values are around 0.0029898366717301225.
```

```
Next, I need to round this value to three decimal places. Looking at
↪  the fourth decimal
  place, which is 8, I round up the third decimal place from 9 to 10,
  ↪  which carries over
  to make it 0.003.

So, the final answer after rounding to three decimal places is 0.003.
</think>

<answer>0.003</answer>
```

R1 (answer-only ablation, same base) — answer 0.124 — ✗ incorrect; trace score 2: unloaded-data call burst with a premature answer, then both branch decisions flipped by misread tiny p-values

```
<think>
Okay, this is a complex question. Let me break it down step by step.
↪  The user wants the last
  non-NaN value of the 30-step trailing rolling-window
  ↪  Bessel-corrected sample variance. But
  to get that, I need to follow several steps.

First, I need to determine which column ('High' or 'Low') has the
↪  larger Wald-Wolfowitz
  runs-test p-value over rows [0, 1459). Then, using that column,
  ↪  check which of the two
  time ranges (rows [0, 1459) or [0, 2919)) gives a larger Engle
  ↪  ARCH-LM test p-value with 5
  lagged squared residuals. Then, using that range, compute the
  ↪  30-step trailing rolling
  variance, and take the last non-NaN value.

So first, I'll run the runs_test_pvalue on both 'High' and 'Low'
↪  columns in the range [0,
  1459). Let's say the higher p-value is from one of them. Then,
  ↪  using that column, check
  the two ranges. For each range, perform the Engle ARCH-LM test with
  ↪  lag 5. The larger
  p-value would determine which range to use. Then, in that selected
  ↪  range, compute the
  rolling variance with window 30, and get the last non-NaN.

I need to call the runs_test_pvalue function for both columns. Then,
↪  based on which is
  higher, proceed. Then, for that column, check the two ranges for
  ↪  the ARCH-LM test. Then,
  based on that, compute the rolling variance. The answer is the last
  ↪  non-NaN in that.
</think>

<tool_call>
{"name": "runs_test_pvalue", "arguments": {"column": "High", "start":
↪  "0", "end": "1459"}}
</tool_call>
<tool_call>
{"name": "runs_test_pvalue", "arguments": {"column": "Low", "start":
↪  "0", "end": "1459"}}
</tool_call>
<tool_call>
{"name": "arch_lm_test_pvalue", "arguments": {"column": "High",
↪  "lag": 5, "start": "0",
  "end": "1459"}}
</tool_call>
<tool_call>
```

```
{"name": "arch_lm_test_pvalue", "arguments": {"column": "High",
↪  "lag": 5, "start": "0",
  "end": "2919"}}
</tool_call>
<tool_call>
{"name": "arch_lm_test_pvalue", "arguments": {"column": "Low", "lag":
↪  5, "start": "0",
  "end": "1459"}}
</tool_call>
<tool_call>
{"name": "arch_lm_test_pvalue", "arguments": {"column": "Low", "lag":
↪  5, "start": "0",
  "end": "2919"}}
</tool_call>
<tool_call>
{"name": "rolling_window", "arguments": {"column": "High", "op":
↪  "var", "window": 30,
  "start": "0", "end": "2919"}}
</tool_call>
<tool_call>
{"name": "rolling_window", "arguments": {"column": "Low", "op":
↪  "var", "window": 30,
  "start": "0", "end": "2919"}}
</tool_call>
<answer>0.123</answer>user
<tool_response>
{"error": "No data loaded. Call load_csv_data first."}
</tool_response>
<tool_response>
{"error": "No data loaded. Call load_csv_data first."}
</tool_response>
<tool_response>
{"error": "No data loaded. Call load_csv_data first."}
</tool_response>
<tool_response>
{"error": "No data loaded. Call load_csv_data first."}
</tool_response>
<tool_response>
{"error": "No data loaded. Call load_csv_data first."}
</tool_response>
<tool_response>
{"error": "No data loaded. Call load_csv_data first."}
</tool_response>
<tool_response>
{"error": "'NoneType' object has no attribute 'iloc'"}
</tool_response>
<tool_response>
{"error": "'NoneType' object has no attribute 'iloc'"}
</tool_response>
assistant
<think>
Okay, the user is asking for a complex analysis on a stock time
↪  series dataset. The previous
  attempts failed because the data wasn't loaded. First, I need to
  ↪  call load_csv_data to
  load the dataset. Then, I need to follow the steps outlined in the
  ↪  question.

The dataset path is given as
  "data/big_company_stock/Amazon/Amazon.csv". So, the first
  step is to call load_csv_data with this file path. Once the data is
  ↪  loaded, I can proceed
  with the subsequent steps.
```

```
Next, I need to determine which column ('High' or 'Low') has the
↪  larger Wald-Wolfowitz
  runs-test p-value over rows [0, 1459). Then, based on that, check
  ↪  which of the two ranges
  ([0, 1459) or [0, 2919)) gives a larger Engle ARCH-LM test p-value
  ↪  with 5 lagged squared
  residuals. Then, using that range, compute the 30-step trailing
  ↪  rolling-window
  Bessel-corrected sample variance, and take the last non-NaN value.

So, the first step is to load the data. Then, run the Wald-Wolfowitz
↪  runs-test on both
  'High' and 'Low' columns. Then, for the selected column, run the
  ↪  ARCH-LM test on both
  ranges. Then, for the selected range, compute the rolling variance.
  ↪  The answer is the last
  non-NaN in that.
</think>

<tool_call>
{"name": "load_csv_data", "arguments": {"file_path":
  "data/big_company_stock/Amazon/Amazon.csv"}}
</tool_call>user
<tool_response>
{"success": true, "shape": [5839, 5], "columns": ["High", "Low",
↪  "Close", "Adj Close",
  "Volume"], "missing_values": {"High": 0, "Low": 0, "Close": 0, "Adj
  ↪  Close": 0, "Volume":
  0}, "timestamp_detected": true, "date_range": {"start": "1970-01-01
  ↪  00:00:00", "end":
  "1970-01-01 00:00:00.000000187"}}
</tool_response>
assistant
<think>
Okay, the data was successfully loaded. Now I need to proceed with
↪  the steps outlined in the
  question. First, I need to find which column ('High' or 'Low') has
  ↪  the larger
  Wald-Wolfowitz runs-test p-value over rows [0, 1459). Then, using
  ↪  that column, check which
  of the two ranges ([0, 1459) or [0, 2919)) gives a larger Engle
  ↪  ARCH-LM test p-value with
  5 lagged squared residuals. Then, in that selected range, compute
  ↪  the 30-step trailing
  rolling-window Bessel-corrected sample variance, and take the last
  ↪  non-NaN value.

So, first, I'll run the Wald-Wolfowitz runs-test on both 'High' and
↪  'Low' columns in the
  range [0, 1459). Let's start with the 'High' column.
</think>

<tool_call>
{"name": "runs_test_pvalue", "arguments": {"column": "High", "start":
↪  "0", "end": "1459"}}
</tool_call>
<tool_call>
{"name": "runs_test_pvalue", "arguments": {"column": "Low", "start":
↪  "0", "end": "1459"}}
</tool_call>
<tool_call>
{"name": "arch_lm_test_pvalue", "arguments": {"column": "High",
↪  "lag": 5, "start": "0",
  "end": "1459"}}
```

```
</tool_call>
<tool_call>
{"name": "arch_lm_test_pvalue", "arguments": {"column": "High",
↪  "lag": 5, "start": "0",
  "end": "2919"}}
</tool_call>
<tool_call>
{"name": "arch_lm_test_pvalue", "arguments": {"column": "Low", "lag":
↪  5, "start": "0",
  "end": "1459"}}
</tool_call>
<tool_call>
{"name": "arch_lm_test_pvalue", "arguments": {"column": "Low", "lag":
↪  5, "start": "0",
  "end": "2919"}}
</tool_call>user
<tool_response>
{"value": 1.3527559046617671e-300}
</tool_response>
<tool_response>
{"value": 6.956040358998316e-294}
</tool_response>
<tool_response>
{"value": 4.0618126019601845e-308}
</tool_response>
<tool_response>
{"value": 0.0}
</tool_response>
<tool_response>
{"value": 6.262231687836997e-308}
</tool_response>
<tool_response>
{"value": 0.0}
</tool_response>
assistant
<think>
Okay, let's process the results. The Wald-Wolfowitz runs-test
↪  p-values for 'High' and 'Low'
  in [0, 1459) are 1.35e-300 and 6.96e-294, respectively. The higher
  ↪  is 'High'. Now, for
  'High', the ARCH-LM p-values in [0, 1459) and [0, 2919) are
  ↪  4.06e-308 and 0.0. The larger
  is 0.0 (from [0, 2919)). So, we use that range. Now, compute the
  ↪  30-step trailing rolling
  variance on 'High' in [0, 2919), take the last non-NaN.
</think>

<tool_call>
{"name": "rolling_window", "arguments": {"column": "High", "op":
↪  "var", "window": 30,
  "start": "0", "end": "2919"}}
</tool_call>user
<tool_response>
{"value": [0.1426603015229712, 0.12126582153630537,
↪  0.09200672478366595, 0.0916715572836144,
  0.09277787295763894, 0.09550596175677482, 0.09553416971691722,
  ↪  0.09513226403005542,
  0.09424535589126344, 0.09424535589126344, 0.0940284091805546,
  ↪  0.09078759966072579,
  0.09349525958489312, 0.09604691592725058, 0.09309544303937427,
  ↪  0.09672389258788129,
  0.10136600068032987, 0.10179999805919647, 0.10127831721719074,
  ↪  0.10110056871768834,
```

```
  0.09875258081888091, 0.10267755650645503, 0.10522202036249399,
  ↪  0.08918168645893912,
  ...(truncated)...97634, 0.14028061882039697, 0.17855502946203425,
  ↪  0.21517748713249568,
  0.2469381822272952, 0.2704578092372792, 0.2961493891771639,
  ↪  0.3168428319556113,
  0.3296236263720889, 0.3360035172094952, 0.345092761033149,
  ↪  0.35614496961827813,
  0.3526424727623073, 0.3516001480771991, 0.340607006099779,
  ↪  0.3208932601546742,
  0.327872663563546, 0.32893156965832876, 0.3032374704184104,
  ↪  0.2748202262073547,
  0.23358823667234133, 0.20371730151223377, 0.1759199899965181,
  ↪  0.14546983960382606,
  0.1291788583251587, 0.1243203232633825]}
</tool_response>
assistant
<think>
Okay, the last element in the rolling variance array is
↪  0.1243203232633825. So, the answer
  is 0.124.
</think>

<answer>0.124</answer>
```

I REPRODUCIBILITY

The RITS dataset with its reference traces, the generation and verification pipeline, the registry-defined tool space, the evaluation and scoring code, and the trained policy checkpoints will be released publicly upon publication. Run configurations are listed in Appendix G.2, the reward instantiation in Appendix F, and retriever details in Appendix E.