VeriGraph: Towards Verifiable Data-Analytic Agents

Jiajie Jin[*], **Zhao Yang**[*], **Wenle Liao, Yuyang Hu,**
Guanting Dong, **Xiaoxi Li**, **Yutao Zhu**, **Zhicheng Dou**[†]
Gaoling School of Artificial Intelligence, Renmin University of China
{jinjiajie, dou}@ruc.edu.cn

Abstract

LLM-based agents have demonstrated strong capabilities in data-intensive analytical tasks, yet their outputs are rarely *verifiable*: a reliance on linear text trajectories makes their reasoning difficult to audit. In particular, deterministic computations over raw data and semantic deductions over natural-language claims are often entangled in an unstructured stream, leaving numerical conclusions hard to reproduce and qualitative judgments hard to inspect. To address this, we propose VeriGraph, a traceable neuro-symbolic reasoning framework that enables agents to construct an explicit heterogeneous evidence directed acyclic graph (DAG) during execution. VeriGraph introduces three evidence-expansion primitives, namely computational, grounding, and derivational expansion, to connect raw data, interpreter variables, computed results, and natural-language claims in a unified graph. Under this formulation, structural traceability is reduced to graph reachability from raw data sources to terminal claims, while semantic support is measured by claim-level evidence evaluation. To improve graph construction, we further design a graph-based policy optimization strategy with a composite reward that jointly supervises answer correctness, computational integrity, and derivational coherence. Experiments on four benchmarks show that VeriGraph-8B achieves the highest overall score among all baselines. More importantly, VeriGraph produces auditable evidence graphs with substantially stronger claim grounding, achieving a 87.61% Grounding Rate under our claim-level evidence support evaluation. These results suggest that explicit evidence-graph construction is a promising path toward *verifiable data-analytic agents*. Our code is available at https://github.com/RUC-NLPIR/VeriGraph.

1 Introduction

Large Language Model (LLM)-based agents [55, 32] have recently demonstrated strong capabilities in tool use [49, 46, 39], code generation [18, 19], and multi-step reasoning [21, 27]. A particularly impactful application of these capabilities is the *data-analytic agent*, which couples an LLM with a code interpreter to tackle data-intensive analytical tasks such as financial analysis [22, 20] and data science [38, 52]. In such tasks, however, trustworthy generation requires more than final-answer accuracy [22, 51, 30]: outputs must be *verifiable*, i.e., users must be able to check how each conclusion is obtained, especially when the answer depends on external data, numerical computation, and multi-step interpretation [60, 15]. For instance, a claim like "Q3 revenue grew 12.3% year-over-year" is trustworthy only when the number is computed from the underlying transaction tables rather than asserted by the model, and that computation is exposed for the reader to verify. This entails two evidence requirements. Quantitative claims must be reproducible from raw data through deterministic computations, and qualitative judgments must be grounded in inspectable reasoning chains.

[*]Equal contribution.
[†]Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

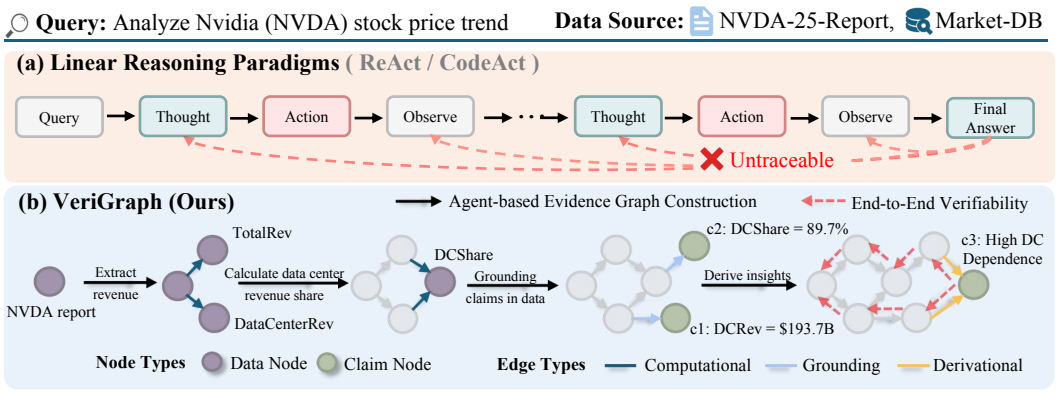


Figure 1: Comparison between our proposed VeriGraph and linear reasoning paradigms.

Current agent paradigms provide little support for this requirement. As illustrated in Figure 1, prevailing frameworks [49, 46] solve problems through a linear trajectory of thought–action–observation steps and ultimately expose only a final answer. This linear transcript entangles two forms of evidence that should be tracked separately. First, intermediate computational artifacts (e.g., interpreter variables) appear only as transient observations, so numerical claims lose their programmatically recoverable provenance [13, 51]. Second, the semantic steps that transform computed values into higher-level judgments remain in free-form text, making supported reasoning difficult to distinguish from confabulation [16, 3, 11, 31]. Recent data-agent work [38, 51, 48] improves end-task accuracy within this paradigm, but still leaves evidence construction outside the learning objective.

Our key observation is that data-intensive reasoning naturally spans two coupled spaces: a *deterministic code space*, where raw data undergo numerical transformations, and a *semantic reasoning space*, where computed results are interpreted and synthesized into higher-level judgments. The provenance crossing these spaces is better represented as a DAG than as a linear transcript. Motivated by this view, we propose VeriGraph (**Veri**fiable Evidence **Graph**), a traceable neuro-symbolic reasoning framework that reformulates the agent's objective from emitting an unstructured text stream to incrementally constructing an explicit *heterogeneous evidence DAG*. Its *data nodes* preserve executable provenance over interpreter variables and computed results, while its *claim nodes* expose semantic derivations among natural-language facts and judgments.

To construct the graph during execution, VeriGraph introduces three reasoning primitives aligned with the graph's expansion modes: *computational expansion*, which automatically traces variable dependencies in executed code; *grounding expansion*, which anchors a computed value as an atomic claim with deterministic, re-executable provenance; and *derivational expansion*, which derives a new claim from established premises with an explicit justification. These primitives are embedded directly in the agent's code action space, allowing computation and evidence construction to proceed in one interaction loop. Under this formulation, traceability reduces to graph reachability (§3.2): a conclusion is verifiable if and only if every constituent claim can be traced backward through the graph to raw data sources. Rather than claiming to completely eliminate hallucinations, VeriGraph makes the reasoning topology explicit so that failures can be localized to the precise unsupported computation, grounding, or derivation step.

Training then focuses on making graph construction reliable rather than merely imitating trajectories. We use synthesized graph-augmented trajectories only as a cold-start stage, teaching the model the basic syntax of the expansion primitives before policy learning. The central challenge is graph-level credit assignment: outcome-only rewards judge the final answer but ignore whether it is supported by connected computations and justified derivations. We therefore introduce *graph-based policy optimization*, whose composite reward mirrors the graph's layered architecture and decomposes credit across answer correctness, computational integrity, and derivational coherence (§3.4).

We evaluate VeriGraph on four data-intensive benchmarks, including TableBench [47], InfiAgent-DABench [14], DSBench [23], and DAB-Step Research [8]. VeriGraph-8B achieves the highest Overall score among evaluated baselines while producing explicit evidence graphs, suggesting the effectiveness of structured evidence construction for data-intensive reasoning. We evaluate

verifiability on a second axis with *Grounding Rate (GR)*, which decomposes each answer into atomic claims and measures what fraction can be recovered from the evidence artifact exposed by the method.

The core contributions of this paper are summarized as follows:

- **A traceable neuro-symbolic reasoning framework with an explicit evidence graph.** We propose VeriGraph, which externalizes an LLM agent's implicit reasoning into an executable heterogeneous evidence DAG via computational, grounding, and derivational expansion primitives, enabling conclusions to be traced back to raw data and deterministic computations.
- **Graph-based policy optimization for auditable reasoning.** We identify graph-level credit assignment as the key training challenge and design a composite reward whose terms mirror the graph's layered architecture, jointly supervising answer correctness, computational integrity, and semantic coherence of derivational edges.
- **Comprehensive experimental validation.** On TableBench, DSBench, InfiAgent-DABench, and DAB-Step Research, our 8B VeriGraph achieves the highest Overall score among all baselines. We further show through grounding analysis and ablations that both graph-structured primitives and graph-aware rewards are essential to traceability and accuracy gains.

2 Related Work

LLM Agents for Data-Intensive Reasoning. LLM agents equipped with code interpreters or SQL engines are widely used for data-intensive analysis [49, 46, 33]. Existing work improves them by redesigning agent pipelines [12, 29, 52, 13] or scaling task-specific training, from tabular fine-tuning [47, 48, 44] to trajectory synthesis [38, 51] on various benchmarks [47, 23]. However, these systems largely optimize the accuracy of the final-answer on flat trajectories [49, 46, 38, 51], leaving implicit the links between code variables and supported claims. Consequently, even strong data agents [13, 51] remain difficult to audit. VeriGraph addresses this traceability gap by constructing a DAG of executable evidence rather than a linear transcript.

Verifiable Generation and Structural Reasoning. Trustworthy-generation work mainly pursues textual attribution and post-hoc verification, including citation or evidence grounding [11, 31, 26] and retrieval mechanisms [36, 1, 34, 6]. These methods improve semantic support, but typically treat evidence as text spans rather than deterministic computations with variable provenance. Structural reasoning systems compile problems into programs or symbolic queries [10, 2, 17, 53], while verifier-based methods add stronger checks [3, 7]. Closest to our motivation, recent graph-based frameworks evaluate tool-agent trajectories beyond final-answer matching [24], verify reasoning through DAG node blocks [9], or reinforce medical reasoning with critical evidence graphs [35]. These works contextualize the value of graph structure, but mostly use graphs to evaluate or reward reasoning paths in general tool-use or domain-specific settings. VeriGraph instead makes a heterogeneous evidence DAG the online action interface for data agents, coupling executable code provenance with grounded and derived claims.

Reinforcement Learning for LLM Agents. Reinforcement learning has become a standard recipe for training LLM agents [4, 40, 58], driving progress in mathematical reasoning [5, 41], search-augmented QA [21, 43, 28], and coding [25, 37]. Data-analytic agents follow the same recipe, optimizing GRPO-style objectives against final-answer correctness over flat trajectories [38, 51, 48, 40, 5]. However, outcome rewards cannot distinguish a correct answer supported by valid evidence from one reached through unsupported claims. Process rewards such as execution correctness [51] or tool-use efficiency [59] help supervise actions, but still ignore graph-level provenance. VeriGraph lifts reward design to the evidence graph, rewarding answer correctness together with raw-data connectivity and local derivation validity.

3 Methodology

3.1 Problem Formulation

We consider data-intensive analytical tasks specified by a tuple $(q, \mathcal{F}, d)$, where q is a user query, $\mathcal{F} = \{f_1, \ldots, f_n\}$ is a set of heterogeneous data sources (e.g., CSV tables or databases), and d

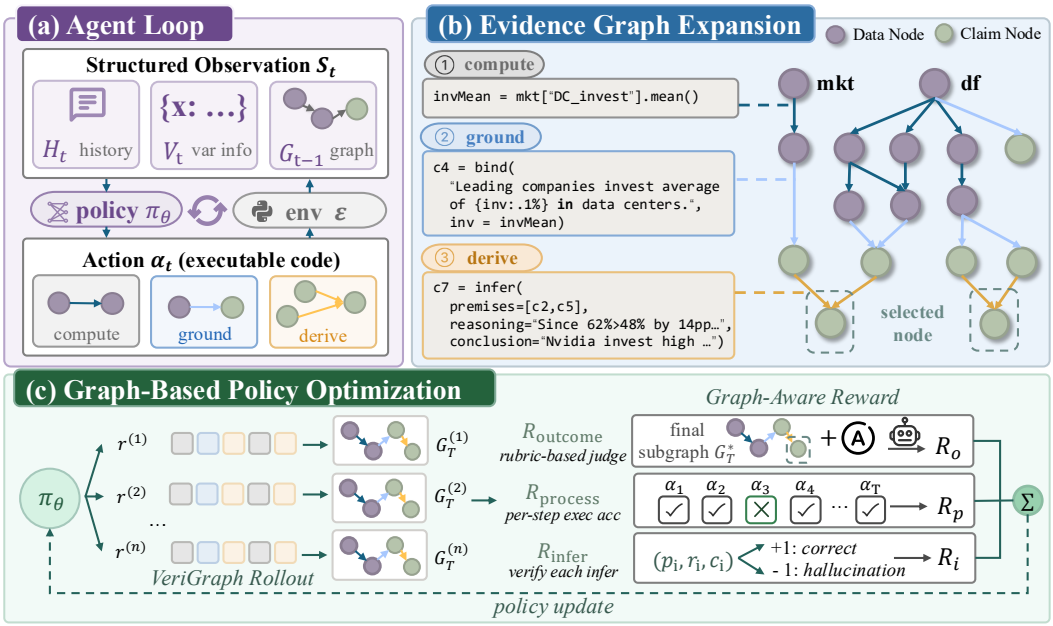


Figure 2: Overview of the VeriGraph framework: The agent iteratively generates code to construct a heterogeneous evidence DAG, optimized via a graph-aware composite reward.

denotes task-specific metadata. Given a ground-truth answer a^*, the goal is to produce a response o that is not only accurate, but also auditable: every factual or numerical claim in o should be supported by evidence ultimately originating from $\mathcal{F}$.

Standard data agents instantiate this task as a ReAct-style interaction [49, 46] with a stateful execution environment $\mathcal{E}$ (e.g., a Python interpreter). At step t, the agent emits a natural-language *Thought* τ_t and a code *Action* α_t, then receives an *Observation* $z_t = \mathcal{E}(\alpha_t)$. The interaction history to step t is:

$$h_t = \big(q, \mathcal{F}, d,\ \tau_0, \alpha_0, z_0,\ \ldots,\ \tau_{t-1}, \alpha_{t-1}, z_{t-1}\big), \tag{1}$$

and the agent samples $(\tau_t, \alpha_t) \sim \pi_\theta(\cdot \mid h_t)$ until termination or a maximum of T rounds. We retain this interaction interface, but reinterpret the rollout as incrementally constructing an evidence graph $\mathcal{G}$. Structural traceability requires support paths from the claims in o to the raw sources in $\mathcal{F}$; semantic support and answer correctness are evaluated separately.

3.2 Overview of VeriGraph

As illustrated in Figure 2(a)–(b), VeriGraph couples a standard agent loop with explicit graph maintenance: each action observes the current history, namespace, and partial graph, then expands the evidence DAG while solving the task. We instantiate this evolving structure as a heterogeneous DAG and formalize auditing as tracing terminal claims back to raw data.

We define the evidence graph as a DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ whose vertex and edge sets are each partitioned into heterogeneous types:

$$\mathcal{V} = \mathcal{V}_{\text{data}} \cup \mathcal{V}_{\text{claim}}, \qquad \mathcal{E} = \mathcal{E}_{\text{comp}} \cup \mathcal{E}_{\text{ground}} \cup \mathcal{E}_{\text{derive}}. \tag{2}$$

Here, $\mathcal{V}_{\text{data}}$ contains raw sources and intermediate computational artifacts, while $\mathcal{V}_{\text{claim}}$ contains natural-language claims. The three edge types capture the provenance relations used in construction: $\mathcal{E}_{\text{comp}} \subseteq \mathcal{V}_{\text{data}} \times \mathcal{V}_{\text{data}}$ records computational dependencies, $\mathcal{E}_{\text{ground}} \subseteq \mathcal{V}_{\text{data}} \times \mathcal{V}_{\text{claim}}$ grounds artifacts into atomic claims, and $\mathcal{E}_{\text{derive}} \subseteq \mathcal{V}_{\text{claim}} \times \mathcal{V}_{\text{claim}}$ records semantic derivations.

3.3 Evidence Graph Construction

Figure 2(b) illustrates one graph-expansion step. At each step t, the agent conditions on a structured observation $S_t = (H_t, \mathcal{V}_t, \mathcal{G}_{t-1})$, where H_t is compressed recent history, $\mathcal{V}_t$ is the interpreter

namespace, and $\mathcal{G}_{t-1}$ is the current graph, and produces an action that simultaneously advances the task and extends the graph:

$$(\tau_t, \alpha_t) \sim \pi_\theta(\cdot \mid S_t), \quad \mathcal{G}_t = \mathcal{G}_{t-1} \cup \underbrace{\Delta\mathcal{G}_t^{\text{comp}}}_{\text{auto-traced}} \cup \underbrace{\Delta\mathcal{G}_t^{\text{ground}} \cup \Delta\mathcal{G}_t^{\text{derive}}}_{\text{agent-invoked}}, \tag{3}$$

where $\Delta\mathcal{G}_t^{\text{comp}}$ is extracted automatically from execution, while $\Delta\mathcal{G}_t^{\text{ground}}$ and $\Delta\mathcal{G}_t^{\text{derive}}$ are created by agent-invoked primitives embedded in α_t: computational expansion tracks how artifacts are produced, `bind` states what an artifact means, and `infer` records how the agent reasons over those meanings. Full rollout pseudocode is provided in Appendix A.1.

Computational Expansion. Executing code automatically expands the code space. Let $\text{NewVars}(\mathcal{E}, \alpha_t)$ denote the set of variables created or modified by action α_t, and $\text{Deps}(v, \alpha_t)$ the variables read to compute v. The computational subgraph update is:

$$\Delta\mathcal{G}_t^{\text{comp}} = \Big(\text{NewVars}(\mathcal{E}, \alpha_t), \ \big\{(u, v) \mid v \in \text{NewVars}(\mathcal{E}, \alpha_t), \ u \in \text{Deps}(v, \alpha_t)\big\} \Big). \tag{4}$$

In practice, NewVars comes from pre/post namespace snapshots and Deps from a static AST walk over α_t (no `sys.settrace`; Appendix A.2).

Grounding Expansion. The `bind` primitive externalizes the semantic interpretation of an executable artifact. It grounds a runtime variable as an atomic claim whose content is expressed in natural language. Given a data node $v_d \in \mathcal{V}_{\text{data}}$ and a natural-language description l of its value:

$$v_c = \texttt{bind}\big(v_d, \ l\big), \quad \Delta\mathcal{G}_t^{\text{ground}} = \big(\{v_c\}, \ \{(v_d, v_c)\}\big), \tag{5}$$

where $v_c \in \mathcal{V}_{\text{claim}}$ is a new atomic claim node. Because `bind` can be applied only to existing artifacts, every grounded claim is anchored to executable evidence. This runtime check enforces provenance, not semantic truth: a misleading description of an existing artifact would still be exposed as a specific grounding edge that must be inspected by the downstream judge or auditor.

Derivational Expansion. The `infer` primitive externalizes the agent's natural-language reasoning over established claims. Rather than introducing new executable evidence, it explicitly records how a higher-level conclusion is derived from a set of premises. Given premises $\mathcal{P} \subseteq \mathcal{V}_{\text{claim}}$, a reasoning annotation r, and a derived conclusion c:

$$v_{\text{new}} = \texttt{infer}\big(\mathcal{P}, \ r, \ c\big), \quad \Delta\mathcal{G}_t^{\text{derive}} = \Big(\{v_{\text{new}}\}, \ \big\{(p, v_{\text{new}}) \mid p \in \mathcal{P}\big\}\Big). \tag{6}$$

This renders each derivational step explicit and auditable.

Terminal Extraction. Generation concludes when the agent invokes a dedicated `submit_answer(`$\mathcal{V}_{\text{final}}$`)` primitive that designates a subset of established claims as terminal nodes:

$$\mathcal{V}_{\text{final}} \subseteq \mathcal{V}_{\text{claim}}, \quad \mathcal{G}^* = \text{Ancestors}_{\mathcal{G}}(\mathcal{V}_{\text{final}}), \quad o = \text{Compose}(\mathcal{V}_{\text{final}}, \ \mathcal{G}^*). \tag{7}$$

The ancestor subgraph $\mathcal{G}^*$ induced by backward traversal from $\mathcal{V}_{\text{final}}$ retains only the nodes and edges supporting the final response, yielding a compact evidence chain. Because `bind` references only existing variables and `infer` only prior claims, $\mathcal{G}$ remains acyclic throughout execution.

3.4 Graph-Based Policy Optimization

Given the evidence-graph construction process above, the central training question is how to make the agent learn not only to answer the task, but also to leave behind a faithful support graph. Standard outcome-supervised RL provides only a terminal signal, e.g., whether the final answer is correct. For our setting, this signal is under-specified: the intermediate objects that determine traceability are evaluated only indirectly. Therefore, a rollout may receive similar final-answer feedback despite failed executions, unsupported semantic jumps, or incomplete provenance chains, making it difficult to assign credit to graph-building decisions that actually determine evidence quality.

We address this with graph-based policy optimization, which aligns reward assignment with the order in which the graph is constructed, as illustrated in Figure 2(c). For each rollout that produces

Table 1: Performance on four data-intensive benchmarks. **Traceability** reflects an output's structural capacity for evidence tracing: **Comp.** (computational provenance) and **Deriv.** (derivational provenance). **GR** measures claim support recoverable from the method's exposed evidence artifact.

Method	**Traceability**		DABench	DSBench	TableBench				DABStep-R		**Overall**	**GR**
	Comp.	Deriv.			Fact	Num.	Anal.	Avg.	Content	Format		
Direct inference												
GPT-5.2	✗	✗	19.46	14.72	85.42	77.83	48.40	66.63	4.61	4.81	48.75	–
Gemini-2.5-Pro	✗	✗	31.91	27.61	82.29	82.62	45.19	67.22	4.82	4.85	55.86	–
Claude-4.5-Sonnet	✗	✗	32.30	25.77	87.50	85.39	53.64	72.61	4.80	4.64	56.27	–
Claude-4.5-Opus	✗	✗	36.58	30.67	84.38	87.41	55.69	74.04	4.68	4.73	58.85	–
Qwen3-32B	✗	✗	22.57	29.75	88.54	84.13	48.40	69.98	4.65	4.64	53.80	–
Qwen3-30B-A3B	✗	✗	26.46	26.69	84.38	85.89	46.06	69.38	4.49	4.53	53.18	–
ReAct data agents												
GPT-5.2	✓	✗	80.93	44.48	75.00	79.09	52.48	67.70	3.26	3.42	64.98	74.17
GPT-5.4	✓	✗	87.16	53.07	77.08	84.63	62.10	74.52	3.71	3.62	72.01	<u>78.52</u>
Claude-4.5-Sonnet	✓	✗	86.77	44.79	72.92	72.07	43.01	62.52	4.55	4.51	71.17	74.16
Claude-4.5-Opus	✓	✗	89.49	53.07	86.15	86.46	57.43	74.40	3.67	3.92	<u>73.22</u>	73.57
Qwen3-8B	✓	✗	80.54	38.04	60.42	72.54	52.19	62.80	3.39	3.41	62.35	59.52
Qwen3-Coder-30B	✓	✗	58.37	28.53	65.62	60.15	35.86	50.78	3.21	3.52	51.25	73.86
QwQ-32B	✓	✗	80.54	46.01	77.08	70.03	51.31	63.16	3.20	3.20	63.43	69.89
Qwen3-32B	✓	✗	84.82	56.44	78.12	79.09	50.73	67.34	3.81	3.72	70.98	71.95
Qwen3-30B-A3B	✓	✗	83.66	47.24	68.75	72.54	51.60	63.52	3.09	3.21	64.36	64.08
Specialized data agents												
DataMind	✓	✗	83.27	14.72	73.96	82.12	50.73	68.94	2.47	2.86	55.06	71.64
DeepAnalyze	✓	✗	75.60	47.64	80.21	71.92	50.40	67.51	3.61	3.93	66.54	61.35
VeriGraph-8B	✓	✓	85.99	66.43	87.23	83.63	57.78	73.58	3.31	3.56	**73.68**	**87.61**

$\mathcal{G}_T$, training observes three objects created by the policy: the action sequence, the derivational expansions, and the extracted terminal evidence subgraph. We attach rewards to these objects at three corresponding granularities: action-level execution rewards supervise the computational backbone, edge-level verification rewards supervise `infer` expansions, and a terminal outcome reward supervises whether the selected evidence subgraph supports the final answer.

Cold-Start via Trajectory Distillation. Base models are pretrained with result-oriented objectives and cannot natively emit the intermediate evidence graphs our framework requires, leaving the RL reward signal too sparse to optimize from such an initialization. We therefore introduce a cold-start stage that distills trajectories from a strong teacher operating inside the full VeriGraph runtime, yielding roughly 36K supervised examples after rejection sampling and rule-based filtering.

We organize these examples at two complementary granularities. The first granularity consists of *atomic samples*, each isolating a single primitive that the agent must master, including next-action prediction under compressed observations, first-step planning from the user query, and final report generation from a completed evidence subgraph. The second granularity consists of *full trajectories*, which preserve the end-to-end construction of an evidence graph and expose the model to long-horizon dependencies. To keep the training distribution faithful to inference, each retained trajectory is replayed through the runtime so that its intermediate results are recovered under the same sliding-window compression used at deployment.

Training proceeds as a two-stage curriculum. The atomic stage shuffles the three types of samples so that the model internalizes the syntax and post-conditions of each primitive without entanglement. The trajectory stage then composes these primitives into coherent multi-turn graph construction. This separation also enables the ablation in §4.5 to attribute gains to the two stages independently. Further details on data construction and training are provided in Appendices B.1.1 and B.2.

Graph-Aware Reward Design. Concretely, the three rewards supervise progressively coarser objects produced by the same rollout: actions $\{\alpha_t\}_{t=1}^{T}$, derivational expansions $\mathcal{I} = \{(\mathcal{P}_i, r_i, c_i)\}_i$,

and the terminal evidence subgraph $\mathcal{G}^*$ from Eq. 7. We therefore use a composite reward that follows the construction order of the graph itself:

$$R(\tau, \mathcal{G}_T) = \underbrace{R_{\text{process}}(\{\alpha_t\}_{t=1}^{T})}_{\text{computational integrity}} + \underbrace{R_{\text{infer}}(\mathcal{I})}_{\text{derivational validity}} + \underbrace{R_{\text{outcome}}(\mathcal{G}^*, q, a^*)}_{\text{terminal subgraph quality}}. \tag{8}$$

The relation among the three terms is hierarchical: R_{process} keeps the computational backbone executable, which is a prerequisite for valid `bind` operations; R_{infer} then checks whether the resulting claim graph contains justified semantic transitions; R_{outcome} finally scores whether the selected terminal subgraph actually answers the task faithfully. We do not introduce a separate grounding reward: a `bind` call is accepted only when it references an existing runtime artifact, so grounding validity is enforced structurally by the environment.

The *process* term operates at action level and aggregates per-step execution feedback over the computational backbone:

$$R_{\text{process}} = \frac{1}{T} \sum_{t=1}^{T} \mathbb{I}[\texttt{exec}(\alpha_t) = \text{success}]. \tag{9}$$

This term encourages the policy to produce executable code paths and stable intermediate variables, rather than reaching the correct answer through brittle or partially failed traces.

The *inference* term operates at the derivational-edge level. For each `infer` invocation $i = (\mathcal{P}_i, r_i, c_i) \in \mathcal{I}$, an external verifier checks whether conclusion c_i follows from premises $\mathcal{P}_i$ under reasoning annotation r_i:

$$R_{\text{infer}} = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \rho(\texttt{Verify}(q, \mathcal{P}_i, r_i, c_i)), \quad R_{\text{infer}} := 0 \text{ if } \mathcal{I} = \emptyset. \tag{10}$$

The verifier returns $s \in \{-1, +1\}$, mapped to the training reward by $\rho(-1) = -0.5$ and $\rho(+1) = +1$. This penalizes unsupported semantic jumps even when the final answer is correct.

The *outcome* term operates at the graph-answer level. An LLM-as-judge scores the extracted terminal subgraph together with the final answer against a task-specific rubric (Appendix C):

$$R_{\text{outcome}} = \mathbb{I}[\text{terminal extraction}] \cdot \texttt{Judge}(q,\, \mathcal{G}^*,\, a^*) \,/\, S, \tag{11}$$

where $\texttt{Judge}(\cdot)$ evaluates answer correctness, completeness, and faithfulness, and S is the maximum rubric score.

We optimize the policy using DAPO [50], treating $R(\tau, \mathcal{G}_T)$ as the trajectory-level return:

$$\mathcal{J}(\theta) = \mathbb{E}_{(q,\mathcal{F},d)\sim\mathcal{D}} \left[\frac{1}{N} \sum_{n=1}^{N} \sum_{t} \min\left(\frac{\pi_\theta}{\pi_{\theta_{\text{old}}}} \hat{A}^{(n)},\ \text{clip}\left(\frac{\pi_\theta}{\pi_{\theta_{\text{old}}}}, 1-\epsilon_l, 1+\epsilon_h \right) \hat{A}^{(n)} \right) \right], \tag{12}$$

where N is the group size and $\hat{A}^{(n)}$ the group-normalized advantage from $R(\tau^{(n)}, \mathcal{G}_T^{(n)})$; per-tuple verifier overhead is $\mathcal{O}(N(|\mathcal{I}| + 1))$, kept within $\sim 1.3\times$ outcome-only rollout cost via batching and a small dedicated verifier (Appendix B.2).

4 Experiments

4.1 Experimental Setup

Benchmarks and Metrics. We evaluate VeriGraph on four data-intensive benchmarks covering three task types and multiple domains. *Table QA*: **TableBench** [47] (∼700 single-table questions over fact-checking, numerical reasoning, and data-analysis subsets). *Data Analysis*: **InfiAgent-DABench** [14] (257 single-CSV questions) and **DSBench** [23] (466 multi-table tasks with long contexts). *Multi-step Research*: **DAB-Step Research**, a 100-case subset curated from DABstep [8], in which each task jointly reasons over tables and unstructured documentation. Accuracy on QA and data-analysis tasks is judged by an LLM. For research tasks, an LLM judge scores each output on *Content* and *Format*, following prior work [51]. We treat evaluation as two-dimensional: benchmark scores measure answer correctness and completeness, while **Grounding Rate (GR)** measures whether the answer's stated claims are recoverable from the evidence artifact exposed by each method. Detailed protocols are in Appendix C.3. We validate the semantic judgments through human auditing, rule-based checks, and cross-model comparisons (Appendices D.5–D.8).

Variant	Infi.	DS	Table	DAB	Overall	GR	Δ
Full VeriGraph-8B	85.99	66.43	73.58	3.31	73.68	87.61	—
Training recipe							
w/o atomic SFT	82.75	57.28	67.61	3.49	69.34	72.33	−4.34
w/o traj. SFT	26.46	69.63	6.22	2.44	37.78	95.01	−35.90
w/o RL stage	85.10	62.40	71.85	3.12	70.42	85.29	−3.26
outcome-only RL	84.05	65.52	64.62	2.15	65.35	76.46	−8.33
Backbone size							
VeriGraph-4B	81.25	65.91	68.44	2.88	68.28	69.41	−5.40
VeriGraph-14B	88.98	70.16	75.74	3.28	75.52	88.67	+1.84

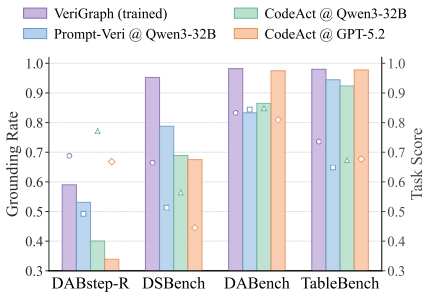


Figure 3: **Left**: ablation on various components and robustness across backbone sizes. **Right**: Grounding Rate vs. task performance across four traceability settings, with task performance shown as points and Grounding Rate shown as bars.

Table 2: Human audit. GR is the percentage of supported claims; pooled GR weights claims across the two datasets. Model labels follow the audit records.

Method	TableBench	DABStep-R	Pooled GR
Claude CodeAct	84.21	17.46	42.57
Qwen CodeAct	40.79	8.67	19.47
VeriGraph	**90.70**	**50.75**	**60.45**

Baselines. We compare against three families of methods: (1) *Direct inference*: LLMs are fed the input files and produce final answers; (2) *ReAct data agents* [49, 46]: equip the agent with a Python tool in a ReAct loop; (3) *Specialized data agents*: data agents trained for data-intensive tasks, including DataMind [38] and DeepAnalyze [51]. See Appendix C.2 for details.

Implementation. Main results use Qwen3-8B [45] as backbone, with at most 50 interaction turns, $8{,}192$ generation tokens, and a $32{,}768$-token context. SFT runs on $36K$ instances with MS-Swift [57] (max sequence length $32{,}768$, learning rate $1\mathrm{e}{-5}$). RL uses Verl [42] with DAPO [50], 8 rollouts per prompt and learning rate $1\mathrm{e}{-6}$. Full details are in Appendix B.2.

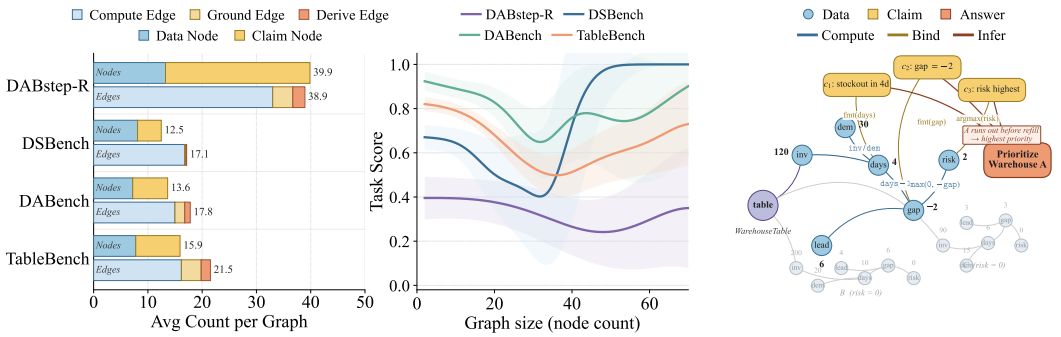
4.2 Main Results

(1) Competitive task performance with explicit evidence. VeriGraph-8B obtains 73.68 Overall, compared with 73.22 for Claude-4.5-Opus ReAct, while exposing both computational and derivational provenance. Their bootstrap intervals overlap (Appendix D.7); we do not claim statistically significant superiority or established equivalence. VeriGraph reaches 87.61 LLM-judged GR, compared with 73.57 for Claude Opus and 78.52 for GPT-5.4 ReAct. Against the same-backbone Qwen3-8B agent, the improvements are $+11.33$ Overall and $+28.09$ GR. **(2) Grounding and report quality measure different properties.** On DAB-Step Research, VeriGraph has strong evidence support but lower Content/Format scores ($3.31/3.56$) than proprietary direct-inference baselines. Explicit provenance does not by itself guarantee complete or well-presented reports; GR is therefore reported alongside task quality.

4.3 Human Audit and Evaluation Robustness

Two non-author annotators assessed whether claims in 180 outputs from TableBench and DAB-Step Research were supported by the exposed evidence. Across 605 claims, VeriGraph has the highest human GR (Table 2). These subset scores differ from benchmark-scale LLM GR and should not be substituted for it. Human–LLM disagreement is larger on research claims; the common ranking supports the comparative grounding result, while absolute scores remain protocol-dependent. Appendix D.5 reports the full breakdown and audit times.

Rule-based answer checks preserve the ranking on TableBench and DABench, and a full-evidence sensitivity study retains the grounding advantage without top-k retrieval (Appendices D.6 and D.8). With CodeAct and Prompt-Veri increased to the same 50-turn budget, VeriGraph retains its lead on



(a) Mean graph composition **(b) Task score vs. graph size** **(c) Warehouse Restocking Case**

Figure 4: Evidence-graph analysis. **(a)** Mean node and edge composition per benchmark. **(b)** Smoothed task score versus graph size with uncertainty bands. **(c)** Example warehouse-restocking DAG with the selected answer chain highlighted.

TableBench and DSBench, while DABench scores are close (Appendix D.9). A 50-case-per-dataset MCTS comparison examines search as an alternative use of inference compute (Appendix D.10).

4.4 Traceability Analysis

Figure 3 (right) isolates the relation between task performance and claim grounding. **(1) Reliable grounding is not a by-product of tool use.** CodeAct agents can reach competitive task scores, yet their GR remains substantially below VeriGraph, showing that a correct-looking trajectory may still leave many final claims hard to audit. **(2) Prompted structure helps but is insufficient.** Prompt-Veri improves over flat CodeAct on grounding, but still trails the trained VeriGraph policy, suggesting that models must learn when and how to materialize evidence rather than only be instructed to output graph-like traces. **(3) VeriGraph improves the accuracy–traceability tradeoff.** It pairs the strongest aggregate GR with strong task performance, supporting explicit evidence-graph construction as a practical route to verifiable data analysis.

4.5 Ablation Studies

Figure 3 (left) ablates the training recipe behind evidence-graph construction. **(1) Trajectory SFT is the enabling stage.** Removing it reduces Overall by 35.90 points and collapses TableBench from 73.58 to 6.22. This indicates that primitive-level imitation can make the model follow the graph interface well, as reflected by its high GR (95.01), but it does not teach the long-horizon composition of code execution, `bind`, and `infer` needed to solve tasks. The result illustrates why GR is evaluated jointly with task performance: locally grounded claims do not imply a complete or useful answer. **(2) Atomic SFT stabilizes the graph interface.** Omitting it yields smaller but consistent drops in Overall (-4.34) and GR (to 72.33), suggesting that short primitive-level examples teach the local preconditions and post-conditions needed for well-typed graph operations. **(3) Graph-aware RL is needed for verifiable gains.** The SFT-only policy remains strong (70.42 Overall, 85.29 GR), while outcome-only RL degrades both accuracy and grounding (65.35 Overall, 76.46 GR). Compared with outcome-only RL, the full policy improves Overall by 8.33 and GR by 11.15 under the same backbone, SFT initialization, data, action space, and training budget. This supports graph-aware supervision as a whole; it does not isolate each reward term or primitive individually.

5 Analysis

5.1 Graph Interpretability

Figure 4 analyzes the terminal evidence graphs produced by VeriGraph. **(1) Graph topology reflects the task regime.** DAB-Step Research yields the largest graphs because many computed signals must be promoted into narrative claims. By contrast, data-analysis benchmarks are dominated by computational edges, while TableBench has a denser claim layer for semantic comparison over tabular facts. **(2) Graph size provides an internal difficulty signal.** Task scores decline as terminal graphs

grow, indicating that larger graphs mainly reflect evidence burden, including additional quantities, joins, comparisons, and cross-source reconciliation, rather than superficial verbosity. **(3) Auditability becomes local.** In the warehouse case, the final recommendation can be traced backward through typed computation, grounding, and derivation edges. This structure localizes potential failures to a specific calculation, grounded statement, or inference step, instead of forcing reviewers to inspect a long linear transcript.

5.2 Robustness Across Backbones

Applying the same recipe to Qwen3-4B and Qwen3-14B (Figure 3, left) shows that the benefit is not tied to a single backbone size. **(1) The graph interface transfers across scales.** Even VeriGraph-4B surpasses the vanilla Qwen3-8B ReAct agent in Overall score, indicating that the gains cannot be explained by capacity alone. **(2) Scaling helps but quickly saturates.** Moving from 4B to 8B yields a larger gain ($+5.40$) than moving from 8B to 14B ($+1.84$), placing the 8B model near a favorable accuracy–cost frontier. **(3) Different tasks stress different bottlenecks.** TableBench continues to improve with model size, whereas DAB-Step Research changes little from 8B to 14B, suggesting that report-style synthesis depends on evidence coverage in addition to raw model capacity.

6 Conclusion

We presented VeriGraph, a neuro-symbolic framework that recasts data-intensive agent reasoning as the incremental construction of a heterogeneous evidence DAG. Through computational, grounding, and derivational expansion primitives embedded in the code action space, VeriGraph unifies deterministic computation with semantic deduction and reduces structural traceability to graph reachability. Built on Qwen3-8B, VeriGraph combines competitive task performance with stronger evidence support under human and model-based evaluation. The benefits carry a task-dependent inference cost, especially on long-horizon tasks. These results suggest that evidence structure should be treated as a first-class optimization target, not merely as a post-hoc explanation attached to an answer.

Acknowledgments and Disclosure of Funding

This work was supported by National Natural Science Foundation of China No. 625B2178 and No. 62272467, Self-Determined Project of the Discipline and Technology Research Center for Large Model Intelligence Applications, Beijing Municipal Science and Technology Project No. Z231100010323009, Beijing Natural Science Foundation No. L233008.

References

[1] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=hSyW5go0v8.

[2] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Trans. Mach. Learn. Res.*, 2023, 2023. URL https://openreview.net/forum?id=YfZ4ZPt8zd.

[3] I-Chun Chern, Steffi Chern, Shiqi Chen, Weizhe Yuan, Kehua Feng, Chunting Zhou, Junxian He, Graham Neubig, and Pengfei Liu. Factool: Factuality detection in generative AI - A tool augmented framework for multi-task and multi-domain scenarios. *CoRR*, abs/2307.13528, 2023. doi: 10.48550/ARXIV.2307.13528. URL https://doi.org/10.48550/arXiv.2307.13528.

[4] Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4299–4307, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/d5e2c0adad503c91f91df240d0cd4e49-Abstract.html.

[5] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, and S. S. Li. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *CoRR*, abs/2501.12948, 2025. doi: 10.48550/ARXIV.2501.12948. URL https://doi.org/10.48550/arXiv.2501.12948.

[6] Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. Chain-of-verification reduces hallucination in large language models. *CoRR*, abs/2309.11495, 2023. doi: 10.48550/ARXIV.2309.11495. URL https://doi.org/10.48550/arXiv.2309.11495.

[7] Haoran Ding, Zhaoguo Wang, and Haibo Chen. FM-Agent: Scaling formal methods to large systems via LLM-based Hoare-style reasoning, 2026. URL https://doi.org/10.48550/arXiv.2604.11556.

[8] Alex Egg, Martin Iglesias Goyanes, Friso Kingma, Andreu Mora, Leandro von Werra, and Thomas Wolf. Dabstep: Data agent benchmark for multi-step reasoning. *CoRR*, abs/2506.23719, 2025. doi: 10.48550/ARXIV.2506.23719. URL https://doi.org/10.48550/arXiv.2506.23719.

[9] Jiwei Fang, Bin Zhang, Changwei Wang, Jin Wan, and Zhiwei Xu. Graph of verification: Structured verification of LLM reasoning with directed acyclic graphs. *CoRR*, abs/2506.12509, 2025. doi: 10.48550/ARXIV.2506.12509. URL https://doi.org/10.48550/arXiv.2506.12509.

[10] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: program-aided language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, Proceedings of Machine Learning Research, pages 10764–10799. PMLR, 2023. URL https://proceedings.mlr.press/v202/gao23f.html.

[11] Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. Enabling large language models to generate text with citations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 6465–6488. Association for Computational Linguistics, 2023. URL https://aclanthology.org/2023.emnlp-main.398.

[12] Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. Ds-agent: Automated data science by empowering large language models with case-based reasoning. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pages 16813–16848. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/guo24b.html.

[13] Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Robert Tang, Xiangtao Lu, Xiawu Zheng, Xinbing Liang, Yaying Fei, Yuheng Cheng, Yongxin Ni, Zhibin Gou, Zongze Xu, Yuyu Luo, and Chenglin

Wu. Data interpreter: An LLM agent for data science. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, Findings of ACL, pages 19796–19821. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.findings-acl.1016/.

[14] Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, Yao Cheng, Jianbo Yuan, Jiwei Li, Kun Kuang, Yang Yang, Hongxia Yang, and Fei Wu. Infiagent-dabench: Evaluating agents on data analysis tasks. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pages 19544–19572. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/hu24s.html.

[15] Wenyue Hua, Xianjun Yang, Mingyu Jin, Zelong Li, Wei Cheng, Ruixiang Tang, and Yongfeng Zhang. Trustagent: Towards safe and trustworthy llm-based agents. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, Findings of ACL, pages 10000–10016. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-EMNLP.585. URL https://doi.org/10.18653/v1/2024.findings-emnlp.585.

[16] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Yejin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12):248:1–248:38, 2023. doi: 10.1145/3571730. URL https://doi.org/10.1145/3571730.

[17] Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. Structgpt: A general framework for large language model to reason over structured data. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 9237–9251. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.574. URL https://doi.org/10.18653/v1/2023.emnlp-main.574.

[18] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation. *CoRR*, abs/2406.00515, 2024. doi: 10.48550/ARXIV.2406.00515. URL https://doi.org/10.48550/arXiv.2406.00515.

[19] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

[20] Antonio Jimeno-Yepes, Yao You, Jan Milczek, Sebastian Laverde, and Renyu Li. Financial report chunking for effective retrieval augmented generation. *CoRR*, abs/2402.05131, 2024. doi: 10.48550/ARXIV.2402.05131. URL https://doi.org/10.48550/arXiv.2402.05131.

[21] Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *CoRR*, abs/2503.09516, 2025. doi: 10.48550/ARXIV.2503.09516. URL https://doi.org/10.48550/arXiv.2503.09516.

[22] Jiajie Jin, Yuyao Zhang, Yimeng Xu, Hongjin Qian, Yutao Zhu, and Zhicheng Dou. Finsight: Towards real-world financial deep research. *CoRR*, abs/2510.16844, 2025. doi: 10.48550/ARXIV.2510.16844. URL https://doi.org/10.48550/arXiv.2510.16844.

[23] Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. DSBench: How far are data science agents from becoming data science experts? In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=DSsSPr0RZJ.

[24] Wonjoong Kim, Sangwu Park, Yeonjun In, Sein Kim, Dongha Lee, and Chanyoung Park. Beyond the final answer: Evaluating the reasoning trajectories of tool-augmented agents. *CoRR*, abs/2510.02837, 2025. doi: 10.48550/ARXIV.2510.02837. URL https://doi.org/10.48550/arXiv.2510.02837.

[25] Chengpeng Li, Mingfeng Xue, Zhenru Zhang, Jiaxi Yang, Beichen Zhang, Bowen Yu, Binyuan Hui, Junyang Lin, Xiang Wang, and Dayiheng Liu. START: self-taught reasoner with tools. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pages 13512–13553. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.EMNLP-MAIN.683. URL https://doi.org/10.18653/v1/2025.emnlp-main.683.

[26] Weitao Li, Junkai Li, Weizhi Ma, and Yang Liu. Citation-enhanced generation for llm-based chatbots. *CoRR*, abs/2402.16063, 2024. doi: 10.48550/ARXIV.2402.16063. URL https://doi.org/10.48550/arXiv.2402.16063.

[27] Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. *CoRR*, abs/2501.05366, 2025. doi: 10.48550/ARXIV.2501.05366. URL https://doi.org/10.48550/arXiv.2501.05366.

[28] Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yutao Zhu, Yongkang Wu, Ji-Rong Wen, and Zhicheng Dou. Webthinker: Empowering large reasoning models with deep research capability. *CoRR*, abs/2504.21776, 2025. doi: 10.48550/ARXIV.2504.21776. URL https://doi.org/10.48550/arXiv.2504.21776.

[29] Ziming Li, Qianbo Zang, David Ma, Jiawei Guo, Tuney Zheng, Minghao Liu, Xinyao Niu, Yue Wang, Jian Yang, Jiaheng Liu, Wanjun Zhong, Wangchunshu Zhou, Wenhao Huang, and Ge Zhang. Autokaggle: A multi-agent framework for autonomous data science competitions. *CoRR*, abs/2410.20424, 2024. doi: 10.48550/ARXIV.2410.20424. URL https://doi.org/10.48550/arXiv.2410.20424.

[30] Robert Litschko, Max Müller-Eberstein, Rob van der Goot, Leon Weber-Genzel, and Barbara Plank. Establishing trustworthiness: Rethinking tasks and model evaluation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 193–203. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.14. URL https://doi.org/10.18653/v1/2023.emnlp-main.14.

[31] Nelson F. Liu, Tianyi Zhang, and Percy Liang. Evaluating verifiability in generative search engines. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 7001–7025. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-EMNLP.467. URL https://doi.org/10.18653/v1/2023.findings-emnlp.467.

[32] Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, Rongcheng Tu, Xiao Luo, Wei Ju, Zhiping Xiao, Yifan Wang, Meng Xiao, Chenwu Liu, Jingyang Yuan, Shichang Zhang, Yiqiao Jin, Fan Zhang, Xian Wu, Hanqing Zhao, Dacheng Tao, Philip S. Yu, and Ming Zhang. Large language model agent: A survey on methodology, applications and challenges. *CoRR*, abs/2503.21460, 2025. doi: 10.48550/ARXIV.2503.21460. URL https://doi.org/10.48550/arXiv.2503.21460.

[33] Peixian Ma, Xialie Zhuang, Chengjin Xu, Xuhui Jiang, Ran Chen, and Jian Guo. SQL-R1: training natural language to SQL reasoning model by reinforcement learning. *CoRR*, abs/2504.08600, 2025. doi: 10.48550/ARXIV.2504.08600. URL https://doi.org/10.48550/arXiv.2504.08600.

[34] Potsawee Manakul, Adian Liusie, and Mark J. F. Gales. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 9004–9017. Association for Computational Linguistics, 2023. URL https://aclanthology.org/2023.emnlp-main.557.

[35] Linjie Mu, Yannian Gu, Zhongzhen Huang, Yakun Zhu, Shaoting Zhang, and Xiaofan Zhang. Medceg: Reinforcing verifiable medical reasoning with critical evidence graph. *CoRR*, abs/2512.13510, 2025. doi: 10.48550/ARXIV.2512.13510. URL https://doi.org/10.48550/arXiv.2512.13510.

[36] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, Findings of ACL, pages 5687–5711. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-EMNLP.378. URL https://doi.org/10.18653/v1/2023.findings-emnlp.378.

[37] Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. *CoRR*, abs/2504.13958, 2025. doi: 10.48550/ARXIV.2504.13958. URL https://doi.org/10.48550/arXiv.2504.13958.

[38] Shuofei Qiao, Yanqiu Zhao, Zhisong Qiu, Xiaobin Wang, Jintian Zhang, Zhao Bin, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Scaling generalist data-analytic agents. *CoRR*, abs/2509.25084, 2025. doi: 10.48550/ARXIV.2509.25084. URL https://doi.org/10.48550/arXiv.2509.25084.

[39] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, 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. *CoRR*, abs/2307.16789, 2023. doi: 10.48550/ARXIV.2307.16789. URL https://doi.org/10.48550/arXiv.2307.16789.

[40] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

[41] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300, 2024. doi: 10.48550/ARXIV.2402.03300. URL https://doi.org/10.48550/arXiv.2402.03300.

[42] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*, pages 1279–1297. ACM, 2025. doi: 10.1145/3689031.3696075. URL https://doi.org/10.1145/3689031.3696075.

[43] Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *CoRR*, abs/2503.05592, 2025. doi: 10.48550/ARXIV.2503.05592. URL https://doi.org/10.48550/arXiv.2503.05592.

[44] Aofeng Su, Aowen Wang, Chao Ye, Chen Zhou, Ga Zhang, Gang Chen, Guangcheng Zhu, Haobo Wang, Haokai Xu, Hao Chen, Haoze Li, Haoxuan Lan, Jiaming Tian, Jing Yuan, Junbo Zhao, Junlin Zhou, Kaizhe Shou, Liangyu Zha, Lin Long, Liyao Li, Pengzuo Wu, Qi Zhang, Qingyi Huang, Saisai Yang, Tao Zhang, Wentao Ye, Wufang Zhu, Xiaomeng Hu, Xijun Gu, Xinjie Sun, Xiang Li, Yuhang Yang, and Zhiqing Xiao. Tablegpt2: A large multimodal model with tabular data integration. *CoRR*, abs/2411.02059, 2024. doi: 10.48550/ARXIV.2411.02059. URL https://doi.org/10.48550/arXiv.2411.02059.

[45] Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

[46] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pages 50208–50232. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/wang24h.html.

[47] Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xeron Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, Tongliang Li, Zhoujun Li, and Guanglin Niu. Tablebench: A comprehensive and complex benchmark for table question answering. In Toby Walsh, Julie Shah, and Zico Kolter, editors, *Thirty-Ninth AAAI Conference on Artificial Intelligence, Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence, Fifteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2025, Philadelphia, PA, USA, February 25 - March 4, 2025*, pages 25497–25506. AAAI Press, 2025. doi: 10.1609/AAAI.V39I24.34739. URL https://doi.org/10.1609/aaai.v39i24.34739.

[48] Zhenhe Wu, Jian Yang, Jiaheng Liu, Xianjie Wu, Changzai Pan, Jie Zhang, Yu Zhao, Shuangyong Song, Yongxiang Li, and Zhoujun Li. Table-r1: Region-based reinforcement learning for table understanding. *CoRR*, abs/2505.12415, 2025. doi: 10.48550/ARXIV.2505.12415. URL https://doi.org/10.48550/arXiv.2505.12415.

[49] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

[50] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Weinan Dai, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: an open-source LLM reinforcement learning system at scale. *CoRR*, abs/2503.14476, 2025. doi: 10.48550/ARXIV.2503.14476. URL https://doi.org/10.48550/arXiv.2503.14476.

[51] Shaolei Zhang, Ju Fan, Meihao Fan, Guoliang Li, and Xiaoyong Du. DeepAnalyze: Agentic large language models for autonomous data science. *CoRR*, abs/2510.16872, 2025. doi: 10.48550/ARXIV.2510.16872. URL https://doi.org/10.48550/arXiv.2510.16872.

[52] Wenqi Zhang, Yongliang Shen, Weiming Lu, and Yueting Zhuang. Data-copilot: Bridging billions of data and humans with autonomous workflow. *CoRR*, abs/2306.07209, 2023. doi: 10.48550/ARXIV.2306.07209. URL https://doi.org/10.48550/arXiv.2306.07209.

[53] Yuyao Zhang, Zhicheng Dou, Xiaoxi Li, Jiajie Jin, Yongkang Wu, Zhonghua Li, Qi Ye, and Ji-Rong Wen. Neuro-symbolic query compiler. *CoRR*, abs/2505.11932, 2025. doi: 10.48550/ARXIV.2505.11932. URL https://doi.org/10.48550/arXiv.2505.11932.

[54] Zhehao Zhang, Xitao Li, Yan Gao, and Jian-Guang Lou. CRT-QA: A dataset of complex reasoning question answering over tabular data. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 2131–2153. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.132. URL https://doi.org/10.18653/v1/2023.emnlp-main.132.

[55] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models. *CoRR*, abs/2303.18223, 2023. doi: 10.48550/ARXIV.2303.18223. URL https://doi.org/10.48550/arXiv.2303.18223.

[56] Yilun Zhao, Yunxiang Li, Chenying Li, and Rui Zhang. Multihiertt: Numerical reasoning over multi hierarchical tabular and textual data. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 6588–6600. Association for Computational Linguistics, 2022. doi: 10.18653/V1/2022.ACL-LONG.454. URL https://doi.org/10.18653/v1/2022.acl-long.454.

[57] Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, Wenmeng Zhou, and Yingda Chen. SWIFT: A scalable lightweight infrastructure for fine-tuning. In Toby Walsh, Julie Shah, and Zico Kolter,

editors, *Thirty-Ninth AAAI Conference on Artificial Intelligence, Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence, Fifteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2025, Philadelphia, PA, USA, February 25 - March 4, 2025*, pages 29733–29735. AAAI Press, 2025. doi: 10.1609/AAAI.V39I28.35383. URL https://doi.org/10.1609/aaai.v39i28.35383.

[58] Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. *CoRR*, abs/2507.18071, 2025. doi: 10.48550/ARXIV.2507.18071. URL https://doi.org/10.48550/arXiv.2507.18071.

[59] Xuhui Zheng, Kang An, Ziliang Wang, Yuhang Wang, and Yichao Wu. Stepsearch: Igniting llms search ability via step-wise proximal policy optimization. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pages 21805–21830. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.EMNLP-MAIN.1106. URL https://doi.org/10.18653/v1/2025.emnlp-main.1106.

[60] Yujia Zhou, Yan Liu, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Zheng Liu, Chaozhuo Li, Zhicheng Dou, Tsung-Yi Ho, and Philip S. Yu. Trustworthiness in retrieval-augmented generation systems: A survey. *CoRR*, abs/2409.10102, 2024. doi: 10.48550/ARXIV.2409.10102. URL https://doi.org/10.48550/arXiv.2409.10102.

[61] Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. TAT-QA: A question answering benchmark on a hybrid of tabular and textual content in finance. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 3277–3287. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.ACL-LONG.254. URL https://doi.org/10.18653/v1/2021.acl-long.254.

Appendix

A VeriGraph Runtime Details . . . 18

A.1 Graph-Augmented Rollout . . . 18

A.2 Runtime Realisation of the Graph Primitives . . . 18

A.3 Per-Turn Observation and Context Compression . . . 20

A.4 RL Integration . . . 20

A.5 Agent System Prompts Used at Training and Inference . . . 20

B Training and Optimization Details . . . 23

B.1 Training Data Construction Details . . . 23

B.1.1 SFT Data . . . 23

B.1.2 RL Data . . . 24

B.2 Training Details . . . 25

C Experimental Protocol . . . 29

C.1 Benchmark Details . . . 29

C.2 Baseline Details . . . 29

C.3 Grounding Rate Evaluation . . . 31

C.4 Evaluation Prompts . . . 32

C.5 Grounding Rate Evaluation Prompts . . . 32

D Additional Analysis . . . 34

D.1 Cost and Latency Analysis . . . 34

D.2 Analysis of Context Management . . . 34

D.3 Failure Modes in RL Training . . . 35

D.4 Cross-Model Consistency Analysis . . . 36

D.5 Independent Human Audit . . . 37

D.6 Rule-Based Evaluation Checks . . . 37

D.7 Bootstrap Uncertainty in Overall Performance . . . 37

D.8 Sensitivity to Evidence Retrieval . . . 38

D.9 Matched Interaction-Budget Baselines . . . 38

D.10 Comparison with Tree-Search Inference . . . 38

E Case Studies . . . 39

E.1 Decision Support over a Tabular Source . . . 39

E.2 Multi-Source Research Report on a Payments Dataset . . . 42

F Reproducibility and Compliance . . . 45

F.1 Compute Resources . . . 45

F.2 Licenses for Existing Assets . . . 45

G Limitations and Future Directions . . . 45

H Statement on the Use of LLMs . . . 46

A VeriGraph Runtime Details

This section complements the formal exposition in Section 3 with a self-contained description of the VeriGraph runtime that is shared by inference, SFT data synthesis, and RL training. Three design points jointly determine the behaviour of the system and are the focus of this appendix: (i) the rollout reinterpreted as graph expansion (Appendix A.1), (ii) the formal semantics of the `bind`, `infer`, and `submit_answer` primitives that the policy invokes inside its code action (Appendix A.2), and (iii) the structured per-turn observation that summarises the executor and graph state under a bounded context budget (Appendix A.3). We close with the integration of this runtime into the RL training loop (Appendix A.4).

A.1 Graph-Augmented Rollout

We adopt the notation of Section 3 without change: q is the user query, $\mathcal{F}$ the attached data sources, π_θ the policy, $\mathcal{E}$ the Python interpreter, $\mathcal{V}_t$ the executor namespace, $\mathcal{G}_t = (\mathcal{V}_{\text{data}} \cup \mathcal{V}_{\text{claim}}, \mathcal{E}_{\text{comp}} \cup \mathcal{E}_{\text{ground}} \cup \mathcal{E}_{\text{derive}})$ the heterogeneous evidence DAG of Eq. (2), and $S_t = (H_t, \mathcal{V}_t, \mathcal{G}_{t-1})$ the structured observation. Algorithm 1 formalises a single rollout. It differs from a vanilla CodeAct loop in three respects: (i) the executor is initialised with a graph runtime exposing the three primitives of Eqs. (5)–(7); (ii) each tool response is rendered together with a structured summary of $\mathcal{V}_t$ and $\mathcal{G}_t$ rather than only `stdout`; and (iii) termination is controlled by the graph state via `submit_answer` rather than by a textual `<answer>` tag.

Algorithm 1 VeriGraph rollout. The policy interacts with a Python interpreter $\mathcal{E}$ endowed with the graph primitives $\Pi = \{\texttt{bind}, \texttt{infer}, \texttt{submit_answer}\}$ and returns the terminal evidence subgraph $\mathcal{G}^*$ together with the set of submitted final claims $\mathcal{V}_{\text{final}} \subseteq \mathcal{V}_{\text{claim}}$.

Require: Query q; files $\mathcal{F}$; policy π_θ; horizon T; recent-window size k; per-field budget b
Ensure: Terminal subgraph $\mathcal{G}^*$ and final claim set $\mathcal{V}_{\text{final}}$

$\mathcal{G}_0 \leftarrow (\emptyset, \emptyset)$, $\mathcal{V}_0 \leftarrow \mathcal{E}.\textsc{Init}(\mathcal{F}, \Pi)$
$H_0 \leftarrow (\textsc{SysPrompt}, q \oplus \mathcal{F})$
for $t = 1, \ldots, T$ **do**
  $S_t \leftarrow (H_{t-1}, \mathcal{V}_{t-1}, \mathcal{G}_{t-1})$ {observation; cf. Sec. 3.3}
  $(\tau_t, \alpha_t) \sim \pi_\theta(\cdot \mid S_t)$ {thought & code action}
  if $\alpha_t = \emptyset$ **then**
    break {degenerate; no graph update}
  end if
  $z_t, \Delta\mathcal{V}_t, \Delta\mathcal{G}_t \leftarrow \mathcal{E}.\textsc{Exec}(\alpha_t)$ {$\Delta\mathcal{G}_t = \Delta\mathcal{G}_t^{\text{comp}} \cup \Delta\mathcal{G}_t^{\text{ground}} \cup \Delta\mathcal{G}_t^{\text{derive}}$, Eq. (3)}
  $\mathcal{V}_t \leftarrow \mathcal{V}_{t-1} \cup \Delta\mathcal{V}_t, \quad \mathcal{G}_t \leftarrow \mathcal{G}_{t-1} \cup \Delta\mathcal{G}_t$
  $\tilde{z}_t \leftarrow \textsc{Render}(z_t, \mathcal{V}_t, \mathcal{G}_t; b)$ {tool response, per-field budget b}
  $H_t \leftarrow \textsc{Compress}\big(H_{t-1} \oplus (\alpha_t, \tilde{z}_t); k\big)$ {recent-k, Eq. (14)}
  if $\textsc{Submitted}(\mathcal{G}_t)$ **then**
    break
  end if
end for
$\mathcal{V}_{\text{final}} \leftarrow \{v \in \mathcal{V}_{\text{claim}} : v.\text{final} = 1\}, \quad \mathcal{G}^* \leftarrow \text{Ancestors}_{\mathcal{G}_t}(\mathcal{V}_{\text{final}})$
return $\mathcal{G}^*, \mathcal{V}_{\text{final}}$

The terminal output o is obtained from $\mathcal{V}_{\text{final}}$ and $\mathcal{G}^*$ via the Compose map of Eq. (7). For report tasks ($|\mathcal{V}_{\text{final}}| > 1$ at inference), Compose is instantiated as a constrained writer that reorders the final claims into prose without introducing new graph nodes, so $\mathcal{G}^*$ remains the sole carrier of evidence. RL rollouts skip this post-hoc writer and optimise directly against $\mathcal{V}_{\text{final}}$.

A.2 Runtime Realisation of the Graph Primitives

The three primitives `bind`, `infer`, and `submit_answer` are defined in Eqs. (5)–(7). Here we specify how the runtime exposes them to the policy and what local checks it applies to keep $\mathcal{G}$ well-formed. We retain all symbols from Section 3: $\mathcal{V}_{\text{data}}$ for data nodes, $\mathcal{V}_{\text{claim}}$ for claim nodes, $\mathcal{V}_{\text{final}}$ for the submitted final claims, and $\mathcal{G}^*$ for the terminal evidence subgraph.

`bind` **(Eq. (5)).** The implementation generalises the single-source signature $v_c = \texttt{bind}(v_d, l)$ to a multi-source one. The agent supplies a templated sentence l containing placeholders $\{x_j\}_{j=1}^{m}$ together with a binding $\{x_j \mapsto v_d^{(j)}\}$ that assigns each placeholder to an existing data node $v_d^{(j)} \in \mathcal{V}_{\text{data}}$. The runtime renders $\text{content}(v_c) = l[\, x_j \mapsto \text{val}(v_d^{(j)})\,]$ and creates the grounding update

$$v_c \in \mathcal{V}_{\text{claim}}, \qquad \Delta\mathcal{G}_t^{\text{ground}} = \Big(\{v_c\},\ \{(v_d^{(j)},\, v_c)\}_{j=1}^{m}\Big), \tag{13}$$

which reduces to Eq. (5) when $m = 1$. A local check rejects templates with no placeholders, so each atomic claim must expose an explicit link to executor state. This check enforces referential anchoring, but does not by itself prove that the surrounding natural-language description is a faithful characterization of the bound values.

`infer` **(Eq. (6)).** The operator $v_{\text{new}} = \texttt{infer}(\mathcal{P}, r, c)$ takes premises $\mathcal{P} \subseteq \mathcal{V}_{\text{claim}}$, a reasoning string r, and a conclusion c, and produces a derived claim $v_{\text{new}} \in \mathcal{V}_{\text{claim}}$ with $\text{content}(v_{\text{new}}) = c$ and $\text{reasoning}(v_{\text{new}}) = r$, inserting the edges of Eq. (6) into $\mathcal{E}_{\text{derive}}$. The runtime rejects non-claim premises, empty c or r, and reasoning strings exceeding a fixed budget, so each $\mathcal{E}_{\text{derive}}$ edge is anchored to existing graph content rather than to free executor state.

`submit_answer` **(Eq. (7)).** $\texttt{submit_answer}(\mathcal{V}_{\text{final}})$ flags each $v \in \mathcal{V}_{\text{final}} \subseteq \mathcal{V}_{\text{claim}}$ as terminal and signals the rollout loop to halt after the current code cell returns. The exporter then materialises the terminal evidence subgraph $\mathcal{G}^* = \text{Ancestors}_{\mathcal{G}}(\mathcal{V}_{\text{final}})$ defined in Eq. (7); subsequent code cells, if any, cannot mutate $\mathcal{G}$.

Provenance and traceability. The graph definition separates computational dependencies, artifact-to-claim attachments, and claim-to-claim derivations. Backward traversal exposes the recorded support of terminal claims. A path to raw data establishes structural traceability under this representation, not the semantic truth of its endpoint. In particular, referring to an existing variable does not establish that the variable was computed faithfully from a raw source, and typing alone is not a proof of source connectivity. The executed cells and artifact snapshots provide the additional context needed for inspection and re-execution. Grounding descriptions and derivations must be assessed separately by a verifier or auditor.

Graph storage and answer extraction. The runtime stores the claim layer $\big(\mathcal{V}_{\text{claim}},\, \mathcal{E}_{\text{ground}} \cup \mathcal{E}_{\text{derive}}\big)$ explicitly as a set of `Claim` records. Each record holds its content, type, and premise ids. For atomic claims it additionally records the bound executor variables $\{x_j \mapsto v_d^{(j)}\}$ and a snapshot of their values at `bind` time. The data layer $(\mathcal{V}_{\text{data}}, \mathcal{E}_{\text{comp}})$ is not mirrored into a separate store. Instead, the runtime keeps the raw inputs loaded at $t = 0$ together with the ordered code cells $(\alpha_1^{\text{code}}, \ldots, \alpha_T^{\text{code}})$ as part of the trajectory. Because the bound identifiers $v_d^{(j)}$ in each atomic claim name specific variables produced by these cells, any data node referenced from the claim layer can be located, inspected, and recomputed by re-executing the prefix that defined it. This supports artifact-level inspection rather than a complete low-level execution trace. Every $v \in \mathcal{V}_{\text{final}}$ reaches data nodes through stored $\mathcal{E}_{\text{ground}} \cup \mathcal{E}_{\text{derive}}$ edges, and each data node is anchored to a specific cell in the executable log. Maintaining a duplicate $\mathcal{E}_{\text{comp}}$ store would have to mirror Python semantics such as pandas mutations, in-place updates, and external library calls to remain accurate, which we found to be a poor engineering trade-off for our analytical workloads. At termination, `submit_answer` triggers an exporter that serialises the claim graph together with $\mathcal{V}_{\text{final}}$, recovers the terminal evidence subgraph $\mathcal{G}^* = \text{Ancestors}_{\mathcal{G}}(\mathcal{V}_{\text{final}})$ by backward closure over premise edges, and reads off the user-facing answer o as a deterministic function of $\mathcal{V}_{\text{final}}$. For QA tasks $|\mathcal{V}_{\text{final}}| = 1$ and o is the content of the single final claim. For report tasks $|\mathcal{V}_{\text{final}}| > 1$ and o is produced by the constrained writer Compose of Eq. (7), which reorders and concatenates $\{\text{content}(v)\}_{v \in \mathcal{V}_{\text{final}}}$ into prose without introducing new claim or data nodes. In both cases any factual content visible to the user is attributable to a node of $\mathcal{G}^*$, which is the invariant on which the traceability metrics of Appendix C.3 rely. RL rollouts skip the report writer and optimise directly against $\mathcal{V}_{\text{final}}$ and $\mathcal{G}^*$.

A.3 Per-Turn Observation and Context Compression

Bounded rendering of S_t. The structured observation $S_t = (H_t, \mathcal{V}_t, \mathcal{G}_{t-1})$ defined in Section 3.3 cannot be passed verbatim, since $\mathcal{V}_t$ may contain large dataframes and $\mathcal{G}_{t-1}$ accumulates over many turns. The runtime therefore exposes S_t to the policy through two deterministic summarisers:

- $\phi_{\text{ns}}(\mathcal{V}_t)$ reports each visible variable by type and shape (e.g., a `DataFrame` as $(\text{rows}, \text{cols})$, an `ndarray` as $(\text{shape}, \text{dtype})$, a scalar by a short preview), so the policy can address $v \in \mathcal{V}_t$ by name without paying for its content.
- $\phi_{\text{cl}}(\mathcal{G}_{t-1})$ lists every $v \in \mathcal{V}_{\text{final}}$ already submitted, plus the most recent K_{cl} ($=30$) non-final claim nodes, each rendered as $(\text{id}, \text{varname}, \text{type}, \text{content}(v))$.

Each field is independently truncated to a per-field budget b ($=1{,}200$ characters) so that no single large object can exhaust the prompt; `stdout` keeps its tail and `stderr` keeps its head, matching the regions most useful for debugging.

Recent-k history compression. Let $|\alpha_t|$ and $|z_t|$ denote the action and raw-observation token counts at step t. The total context after T turns is $L_T = \sum_{t=1}^{T}(|\alpha_t| + |z_t|)$, and in data-analytic settings $\mathbb{E}[|z_t|] \gg \mathbb{E}[|\alpha_t|]$ because z_t may carry table previews and tracebacks, so L_T saturates the context window within only a few turns if left uncompressed. The history component H_t is therefore maintained by a fixed-window compressor that keeps the newest k tool responses in full and replaces older ones with a constant stub:

$$\tilde{z}_{t'} \;=\; \begin{cases} \textsc{Render}(z_{t'},\, \mathcal{V}_{t'},\, \mathcal{G}_{t'};\, b) & t' > t-k \\ \texttt{[omitted tool result]} & t' \le t-k \end{cases} \tag{14}$$

with $k=5$ in the reported configuration. Compression operates on whole tool responses rather than on individual fields, preserving the alignment of the $(\alpha_{t'}, \tilde{z}_{t'})$ pairs that downstream loss-masking relies on. Because persistent facts must be materialised as nodes of $\mathcal{V}_{\text{claim}}$ via `bind` or `infer`, dropping older raw stdouts does not erase evidence: the relevant content has already been promoted into $\mathcal{G}_{t-1}$ and re-enters every subsequent prompt through $\phi_{\text{cl}}(\mathcal{G}_{t-1})$.

A.4 RL Integration

The same runtime is used for RL rollouts. Each RL prompt is paired with a workspace directory and a judge-prompt type derived from the task family (QA versus research), and rollouts are executed inside SGLang with a per-trajectory copy of the executor. Algorithm 1 therefore runs unchanged; the only differences are that (i) the post-hoc report writer is disabled, so the optimisation target is exactly $\mathcal{V}_{\text{final}}$, and (ii) the rollout records, in addition to $\mathcal{G}^*$, a token-level loss mask that selects assistant tokens generated by π_θ and excludes `<tool_response>` content.

Trajectory validity. A trajectory is marked valid if `submit_answer` fired before the turn budget T and at least one tool call succeeded. Invalid trajectories ($\mathcal{V}_{\text{final}} = \emptyset$ or zero successful tool calls) have their loss mask zeroed, so the corresponding tokens contribute neither to the policy gradient nor to the KL term. This protects the policy from imitating its own malformed rollouts during early training.

Composite reward. RL uses the weighted reward detailed in Appendix B.2.

A.5 Agent System Prompts Used at Training and Inference

For completeness and reproducibility, we list the exact system prompts that the VeriGraph policy receives during training and inference. The prompt in Listing 1 is the one used at *evaluation time* and during *single-turn* rollouts (e.g., trajectory synthesis with a strong teacher LLM); it spells out the workflow, the evidence-graph API, the strict adherence rules, and few-shot working examples. The compact prompt in Listing 2 is the system prompt that conditions our 8B policy during *SFT and RL training* as well as during multi-turn inference: it preserves the same workflow and API but removes the few-shot demonstrations, since at training/inference time the agent is already exposed to canonical multi-turn trajectories. Finally, Listing 3 is the system prompt fed to the post-hoc report writer that turns $\mathcal{V}_{\text{final}}$ into prose; this writer is disabled during RL rollouts (Appendix A.4).

Listing 1: VeriGraph single-turn / inference system prompt (VERIGRAPH_PROMPT). Few-shot examples are abbreviated.

```
You are a rigorous Data Analysis Agent operating inside a persistent Python environment.

Your task is to conduct a comprehensive data analysis of the request and file provided by the user,
     and present all useful intermediate analysis conclusions in the form of claims. These claims
     will ultimately serve as material for report writing, so you need to provide as insightful and
     profound claims as possible.

### 1. Core Concept: The Claim System
A Claim is the only persistent unit of knowledge. Standard text you output is ephemeral and will be
     discarded by the system.
- Atomic Claims (bind): Direct, objective observations extracted from Python variables (e.g., "The
     mean weight is 42.5").
- Derived Claims (infer): Logical conclusions or judgments that synthesize multiple existing Claims.
- The Evidence Graph: Data -> bind() -> Atomic Claims -> infer() -> infer() ... -> Final Conclusion.

Each claim must be informative rather than a superficial description of the dataset; you need to
     conduct an in-depth analysis of the data.

### 2. Operational Workflow
You operate in a persistent Python environment. Follow this loop:
1. Analyze/Reasoning: use <think>...</think> to plan steps (loading, cleaning, analysis, synthesis)
     and write your internal reasoning.
2. Act: execute Python code in <code_interpreter>...</code_interpreter>. Whenever you want to express
      information or a conclusion, make claims in code. The environment is persistent; you can reuse
     variables from previous turns.
3. Observe: you will receive output in a <tool_response> block. Verify whether the Claim was
     successfully created and reflects the data.
4. Iterate: repeat the analyze-act-observe loop until you have sufficient claims.
5. Finish: call submit_answer() with the final claims. For QA, submit a single claim. For report
     generation, submit all claims needed by the report.

You have a total of 50 rounds of coding opportunities, so explore and analyze as much as needed.
For open-ended report tasks, plan first and then conduct multi-dimensional in-depth analysis (basic
     statistics as well as statistical methods).

### 3. Evidence Graph API
bind(template_str: str, **kwargs) -> Claim
- Ground objective facts from Python variables. Use {key} placeholders. Keep numeric values to 3
     decimal places.
- No hardcoding: numbers must come from variables (e.g., bind("Mean is {m}", m=df['col'].mean())).
- Use this for user-provided facts: bind("User states X: {x}", x=user_input).
- Record only useful information; do not bind every inspection output.

infer(premises: list[Claim], conclusion: str, reasoning: str) -> Claim
- Use this to express reasoning in natural language when it cannot be expressed through code.
- premises must be a list of previously created Claim objects.
- conclusion must NOT introduce new raw data or facts not in premises; it must be supported by
     premises.
- The conclusion must NOT contain any placeholders.

submit_answer(final_claims: list[Claim] | Claim) -> None
- Finalize the task. Submit the final claim or a list of claims (e.g., [c1, c2, c3_conclusion]).
- Before submit_answer, confirm that the claims you use meet your expectations (see claim content
     from system logs).

### 4. Strict Adherence Rules
- Zero-Text Policy: do not provide answers in plain text; important insights must be Claims.
- Persistence Awareness: reuse variables, imports, DataFrames; do not re-load files unless necessary.
- Traceability: before submit_answer, ensure a human can reconstruct your logic using only the Claims
      you created.
- Verification: never infer based on a Claim you have not seen successfully created in the tool
     response.
- Every time you use infer or submit_answer, first confirm the claims you use meet your expectations.
- Do not create duplicate Claims for the same fact; reuse existing Claim variables as premises when
     possible.
- Context Management: tool outputs are truncated and older outputs may be dropped. Never rely on old
     stdout for important facts; bind them into Claims.

### 5. How To Run Code
Write a Python block exactly like:
<code_interpreter>
# valid python only
</code_interpreter>
Place bind() / infer() / submit_answer() calls inside that Python block.
```

```
Safety: do not write or delete files, run shell commands, or make network requests. Keep prints short
      and only print what you need to decide the next step.

### 6. Working Examples
[Example 1: Open-Ended Research]
User: Analyze 'nhl_data.csv' for player longevity and physical traits.
Assistant: <think> plan inspection, baseline statistics, correlation between physical traits and
     Total_Games, comparative synthesis ... </think>
<code_interpreter>
import pandas as pd
df = pd.read_csv("nhl_data.csv")
n_players = df['player_id'].nunique()
h_stats = df['height'].describe(); w_stats = df['weight'].describe()
c1 = bind("The dataset tracks {n} unique players. Heights {h_min}-{h_max} cm (avg {h_avg:.1f});
     weights {w_min}-{w_max} kg (avg {w_avg:.1f}).",
         n=n_players, h_min=h_stats['min'], h_max=h_stats['max'], h_avg=h_stats['mean'],
         w_min=w_stats['min'], w_max=w_stats['max'], w_avg=w_stats['mean'])
</code_interpreter>
<tool_response> Successfully created claim ... </tool_response>
... [further bind / infer steps elided] ...
<code_interpreter>
c4 = infer(premises=[c1, c2, c3],
           conclusion="In professional hockey, body mass is a better predictor of career longevity
     than height.",
           reasoning="Height correlates weakly (0.12) while weight correlates moderately (0.45); the
     heavy/light gap is ~150 games.")
submit_answer([c1, c2, c3, c4])
</code_interpreter>

[Example 2: Short QA]
User: What is the average BMI of patients in 'patients.csv'?
Assistant: load file, compute BMI = weight / (height/100)^2, take the mean, then
<code_interpreter>
c_bmi = bind("The average BMI across the patient cohort is {avg:.2f} kg/m^2.", avg=avg_bmi)
submit_answer(c_bmi)
</code_interpreter>
```

Listing 2: VeriGraph SFT/RL training and multi-turn inference system prompt (`VERIGRAPH_PROMPT_SFT`).

```
You are a rigorous Data Analysis Agent operating in a persistent Python environment. Your task is to
     analyze data and formulate insights strictly using a Claim-based API.

### 1. Workflow
1. Plan: use <think>...</think> to outline your reasoning and next steps.
2. Execute: write Python code inside <code_interpreter>...</code_interpreter>.
3. Observe: wait for the system's <tool_response>.
4. Submit: once you have sufficient evidence, call submit_answer().

### 2. Evidence Graph API (Available in Python)
You MUST construct all insights using these persistent Claim objects:
- bind(template_str: str, **kwargs) -> Claim: create an atomic claim directly from data variables.
     Format numbers to 3 decimal places. Example: c1 = bind("Mean is {m}", m=val).
- infer(premises: list[Claim], conclusion: str, reasoning: str) -> Claim: derive a logical conclusion
      from existing claims; do not introduce new raw data here.
- submit_answer(final_claims: list[Claim] | Claim) -> None: terminate the task and submit your final
     claims as the answer.

### 3. Strict Constraints
- Zero-Text Policy: standard text outside code blocks is ephemeral. All meaningful insights and final
      answers must be instantiated via the Claim APIs.
- Persistence: the Python environment is persistent. Reuse imported libraries, DataFrames, and
     previously created Claim variables.
- Verification: never pass a Claim to infer or submit_answer until you have verified its successful
     creation in the preceding <tool_response>.
```

Listing 3: Post-hoc report writer prompt (`REPORT_PROMPT`). Disabled during RL rollouts.

```
You are given a user question and a list of final claims.
Write a clean final report in Markdown using only those claims.
Do not invent facts.
```

B Training and Optimization Details

B.1 Training Data Construction Details

We curate the training corpus from a heterogeneous pool of publicly available table-grounded and data-analysis benchmarks, and re-purpose them into evidence-graph trajectories that exercise the primitives defined in Section 3. Throughout this section we denote a trajectory by $\tau = (q, \{(a_t, o_t)\}_{t=1}^{T})$, where q is the task with its associated files, a_t a graph-aware action, and o_t the corresponding observation.

Source datasets. Our training corpus follows the public datasets and data mixtures used in prior table-reasoning and data-agent work [51, 38], rather than introducing new task instances or additional benchmark annotations. We draw on six upstream source families that cover complementary regimes of table-grounded and data-analytic reasoning. The pool consists of TableInstruct, the instruction data released with TableBench [47] ($\sim$19K); TAT-QA [61] ($\sim$13K), a finance QA benchmark over hybrid tabular and textual evidence; CRT-QA [54] (0.7K), which emphasizes complex reasoning over tables; MultiHiertt [56] ($\sim$7K), which requires numerical reasoning over multi-hierarchical tables and associated text; DataScience-Instruct from DeepAnalyze [51], split into a TableQA partition ($\sim$3K) and an open-ended data-analysis partition ($\sim$10K); and DataMind [38] ($\sim$7.3K). Together, these sources span single-table QA, hybrid table–text QA, multi-table and multi-document reasoning, and open-ended data analysis. The single-table QA sources provide short, high-precision supervision for atomic evidence-graph operations, whereas MultiHiertt, DataMind, and the open-ended DataScience-Instruct partition contribute longer trajectories that require cross-file computation, quantitative synthesis, and evidence-preserving report generation.

B.1.1 SFT Data

The SFT data is constructed to teach the model both how to solve data-analysis tasks and how to maintain the evidence graph while solving them. We first synthesize complete task trajectories from the source datasets, then filter them for executable and logically consistent graph construction, and finally derive atomic next-action samples from the retained trajectories.

Trajectory synthesis. For each source example, we use a stronger teacher model (Qwen3-32B) to generate a full solution trajectory. The teacher is run through the same VeriGraph runtime rather than asked to emit a static transcript: it receives the system prompt in Listing 4, calls the Python sandbox with the evidence-graph API, observes the executor feedback, and stops only after calling `submit_answer`. The annotation script uses temperature 0.6, top-p 0.95, top-k 20, repetition penalty 1.1, a maximum of 8,192 generated tokens per model call, 120s per code cell, and 400s per task. We cap each trajectory at 50 interaction turns to avoid unbounded generation while still allowing long multi-step analyses.

Listing 4: Trajectory annotation system prompt used for SFT synthesis. Demonstration examples in the code are omitted for space.

```
You are a rigorous Data Analysis Agent operating inside a persistent Python environment.

Your task is to conduct a comprehensive data analysis of the request and file provided by the user,
     and present all useful intermediate analysis conclusions in the form of claims.
These claims will ultimately serve as material for report writing, so you need to provide as
     insightful and profound claims as possible.

### Core Concept: The Claim System
A Claim is the only persistent unit of knowledge. Standard text you output is ephemeral and will be
     discarded by the system.
- Atomic Claims (bind): Direct, objective observations extracted from Python variables.
- Derived Claims (infer): Logical conclusions or judgments that synthesize multiple existing Claims.
- The Evidence Graph: Data -> bind() -> Atomic Claims -> infer() -> ... -> Final Conclusion.

### Operational Workflow
1. Use <think>...</think> to plan loading, cleaning, analysis, and synthesis.
2. Execute Python code inside <code_interpreter>...</code_interpreter>.
3. Whenever you want to express information or a conclusion, create a claim in Python.
4. Review the <tool_response> to verify whether the claim was successfully created.
5. Repeat until there are sufficient claims, then call submit_answer().

### Evidence Graph API
```

```
bind(template_str: str, **kwargs) -> Claim
- Ground objective facts from Python variables. Keep numeric values to 3 decimals.
- Do not hardcode numbers; values must come from variables.

infer(premises: list[Claim], conclusion: str, reasoning: str) -> Claim
- Derive a natural-language conclusion from existing claims.
- Do not introduce raw data or facts that are not supported by the premises.

submit_answer(final_claims: list[Claim] | Claim) -> None
- Finalize the task. For QA, submit the final claim; for report tasks, submit all claims needed by
    the report.

### Strict Rules
- Zero-Text Policy: all meaningful insights and final answers must be instantiated as Claims.
- Persistence: reuse variables, imports, DataFrames, and existing Claim variables.
- Traceability: before submit_answer, ensure a human can reconstruct the logic using only the created
     Claims.
- Verification: never infer from or submit a Claim until its successful creation appears in the tool
    response.
- Safety: do not write files, delete files, run shell commands, or make network requests.
```

Trajectory filtering. Because synthesized trajectories may contain malformed code, invalid tool-call syntax, evidence-graph violations, or inconsistent reasoning, we apply automatic filters followed by targeted manual inspection. The filtering pipeline removes trajectories that are unsuitable for training, including those with invalid formats, malformed tool calls, missing graph-maintenance actions, no final submitted answer, or unreliable code execution, defined as more than 50% of code cells failing. We also discard trajectories shorter than 256 tokens or longer than 32K tokens, as well as trajectories whose final answers receive a low LLM-as-judge score when compared against the gold answer or task-specific rubric. Rather than retaining only error-free traces, we preserve a small fraction of executable and structurally valid trajectories that contain localized mistakes, failed intermediate attempts, or suboptimal reasoning steps, provided that they are ultimately recoverable and lead to an acceptable final answer. We define recoverable errors as localized mistakes in intermediate steps that do not invalidate the trajectory structure or final answer, and that are corrected or safely bypassed in subsequent steps. This yields a training set that remains structurally reliable and aligned with the expected answer, while exposing the model to controlled error-and-recovery patterns that may arise during problem solving.

Atomic sample construction. From the filtered full trajectories, we construct fine-grained SFT samples with a sliding-window scheme. Given a trajectory with T turns, we keep the prefix up to turn t as context and use the next action at turn $t + 1$ as the prediction target. To control context length and remove redundant observations, the per-turn context is compressed before constructing atomic samples, following the procedure described in Appendix A.3. Repeating this over each trajectory yields multiple atomic training examples, each focused on the next evidence-graph operation or tool action. During atomic sample construction, we retain an example only when the target next action and its resulting feedback pass the validity and correctness checks; if the execution result of the target step is erroneous, the corresponding atomic example is discarded. However, the preceding context is not required to be error-free. When earlier turns contain localized mistakes but the current target step correctly recovers from, corrects, or bypasses them, we preserve those imperfect historical contexts. This allows the model to learn not only standard next-action prediction under clean contexts, but also how to perform self-correction in the presence of prior errors. We keep three atomic task types: first-step planning from the user query and file list, next-action prediction after compressed tool feedback, and final submission/report construction from the completed claim set. These samples complement full-trajectory imitation by directly supervising local graph-maintenance decisions and recovery-oriented actions.

B.1.2 RL Data

RL prompts are drawn from the same pool as the SFT corpus but with strictly non-overlapping queries; the per-source quotas are listed in Table 3. Short single-table QA sources are down-weighted and long multi-document analysis sources are up-weighted, so that the RL stage focuses on prompts that genuinely benefit from multi-step graph construction.

Table 3: SFT and RL data mixture. Counts are approximate.

Source	SFT	RL
DataScience-Instruct (TableQA)	2.0K	2.0K
DataScience-Instruct (open-end)	5.5K	3.7K
CRT-QA	0.5K	0.2K
MultiHiertt	3.1K	0.7K
TableInstruct	12.0K	2.0K
TAT-QA	7.9K	1.3K
DataMind	5.1K	2.1K
Total	**36K**	**12K**

Table 4: Key hyperparameters in SFT phase.

Hyperparameter	Value
Max Sequence length	32768
Training Epochs	2
Effective global batch size	128
Per-device train / eval batch	2 / 4
Gradient accumulation	16
Learning rate	1×10^{-5}
Warm-up ratio	0.05
Precision	bfloat16

Difficulty filtering. To avoid the well-known reward-collapse phenomenon on prompts that are either saturated or unsolvable, we run $K = 8$ rollouts per prompt with the SFT-only policy, estimate the empirical pass rate $\hat{p}$, and keep prompts with $\hat{p} \in [p_{\min}, p_{\max}] = [0.1, 0.8]$. For open-ended tasks, the "pass" label is replaced by a rubric score above threshold from the same LLM-as-judge used in SFT filtering. The classifier prompt used for difficulty annotation is shown in Listing 5.

Listing 5: RL difficulty-classifier prompt (abbreviated).

```
Given a task description, the data files, K rollouts of the SFT policy, and the
gold answer (or rubric), label the task as {trivial, easy, medium, hard,
unsolvable} based on the empirical pass rate and reasoning depth required.
Output JSON: {"pass_rate":..., "depth":..., "label":..., "keep":true|false}.
Keep iff label in {easy, medium, hard} and pass_rate in [0.1, 0.8].
```

Final RL pool. The above procedure produces 12K RL prompts. For fast hyper-parameter sweeps we additionally use a stratified 1K subset that preserves the per-source proportions of the full pool.

B.2 Training Details

SFT. We train the Qwen3-8B backbone with full-parameter SFT using MS-Swift [57]. The corpus contains the 36K examples in Table 3. We first train on the atomic next-action samples and then mix in complete trajectories so that the model learns both the primitive syntax and long-horizon graph construction. Table 4 lists the implementation settings.

RL. We initialize RL from the SFT checkpoint and optimize with DAPO [50] in Verl [42]. The rollout engine is SGLang in multi-turn mode, using the same VeriGraph code executor and evidence-graph runtime as evaluation. Prompts are converted to chat messages with the SFT-time VeriGraph prompt, a user question, and the attached file names; the preprocessor keeps the task directory in `extra_info` so each rollout executes in the correct workspace. Table 5 reports the main composite-reward setting; the fast ablation scripts reduce the rollout group size to 4.

Reward design. The reward shaping signal in our RL loop is produced by a unified verifier module that decomposes the trajectory-level reward into the same three components introduced in the main text (Eq. 8), namely R_{process}, R_{infer}, and R_{outcome}, each targeting a distinct failure mode of long-

Table 5: Key hyperparameters in RL phase.

Category	Hyperparameter	Value
Training	Rollout Backend	SGLang
	Rollouts per prompt	8
	Train prompt batch size	16
	Generation prompt batch size	4
	PPO mini-batch size	2
	PPO micro-batch per GPU	1
	Learning rate	1×10^{-6}
	LR warm-up steps	10
	Weight decay	0.1
	KL loss coefficient	0.01
Rollout	Temperature	1.0
	Top-p	0.95
	Repetition penalty	1.1
	Max response length	8192
	Max interaction turns	50
	Tool observation budget	1,200 chars
	Tool timeout	120s
	Trajectory timeout	1,800s
Reward	Process reward weight	0.40
	Final reward weight	0.35
	Infer reward weight	0.15
	Reward clipping (Min)	-0.3
	Reward clipping (Max)	0.8
	Reward judge model	`gpt-4o-mini`

horizon analytic reasoning. To stabilise optimisation in practice, the implemented trajectory reward extends the main-text composite with per-component weights, a missing-submission penalty, and an overlong-response shaping term:

$$R \;=\; w_{\mathrm{p}}\, R_{\text{process}} \;+\; w_{\mathrm{i}}\, R_{\text{infer}} \;+\; w_{\mathrm{f}}\, R_{\text{outcome}} \;+\; \mathbb{1}[\,\mathcal{V}_{\text{final}} = \emptyset\,] \cdot p_{\text{sub}} \;+\; R_{\text{overlong}}, \tag{15}$$

where $\mathcal{V}_{\text{final}}$ denotes the final claim set submitted by the policy. The weights $(w_{\mathrm{p}}, w_{\mathrm{i}}, w_{\mathrm{f}})$ and the missing-submission penalty p_{sub} are taken from Table 5, R_{overlong} is inherited from DAPO [50], and a final clipping step to $[-0.3, 0.8]$ is applied for variance control. Setting $w_{\mathrm{p}} = w_{\mathrm{i}} = w_{\mathrm{f}} = 1$ and disabling the auxiliary terms recovers the main-text form exactly. Following standard practice for trajectory-level RL with token-level loss masking, the scalar reward is written onto the last assistant-generated token, so that all generated tokens share a single advantage during the policy update. The infer-edge and outcome verifiers are realised by the same off-the-shelf judge model (`gpt-4o-mini` in our experiments), and judge calls are issued asynchronously across the rollout batch under a fixed concurrency budget so as to avoid stalling the rollout–update pipeline.

Process reward. R_{process} is a deterministic, verifier-free signal that captures the syntactic and operational health of the trajectory. Following Eq. 9 in the main text, it is the empirical success rate of the executed actions against the Python sandbox,

$$R_{\text{process}} \;=\; \frac{1}{T} \sum_{t=1}^{T} \mathbb{I}[\texttt{exec}(\alpha_t) = \text{success}],$$

and set to zero whenever the trajectory is marked invalid or contains no tool calls. Because this term is computed from the executor's own feedback, it incurs no additional verifier traffic and can be evaluated densely at no cost. Empirically it serves to stabilise early training, where the dominant failure mode is malformed code rather than substantive reasoning errors.

Inference reward. R_{infer} is the central novelty of our reward design and operates at the granularity of individual derivational edges in the evidence graph. For each edge $(\mathcal{P}_i, v_{\text{new}}) \in \mathcal{E}_{\text{derive}}$ introduced by an `infer` primitive (Eq. 6), the verifier receives the rendered contents of the premise claims $\mathcal{P}_i$, the conclusion $c_i = \text{content}(v_{\text{new}})$, and the natural-language justification $r_i = \text{reasoning}(v_{\text{new}})$, and is asked to decide whether c_i is logically entailed by, or at least clearly licensed by, $\mathcal{P}_i$. The raw verifier

output is $s_i = \texttt{Verify}(q, \mathcal{P}_i, r_i, c_i) \in \{-1, +1\}$: $+1$ indicates an inference faithful to its premises, and -1 an unsupported leap, hallucinated quantity, speculative causal attribution, or contradiction. As in Eq. 10, training maps these labels through $\rho(-1) = -0.5$ and $\rho(+1) = +1$. The per-trajectory inference reward is the mean of the mapped rewards over all derivational edges in $\mathcal{G}^*$, and is defined to be zero when no `infer` edge is present. The exact judge prompt is given in Listing 8. Crucially, this term penalises locally invalid reasoning even when the final answer happens to be numerically correct, and thereby couples optimisation pressure to the structural integrity of the evidence graph rather than to its terminal output alone.

Outcome reward. R_{outcome} instantiates the main-text definition (Eq. 11), $R_{\text{outcome}} = \mathbb{I}[\text{terminal extraction}] \cdot \texttt{Judge}(q, \mathcal{G}^*, a^*)/S$, and measures the quality of the submitted answer relative to the task specification. It is computed only for valid trajectories that terminate via `submit_answer` with a non-empty $\mathcal{V}_{\text{final}}$; trajectories that fail to submit incur the fixed penalty p_{sub} in Eq. 15 so that the verifier is never invoked on degenerate rollouts. To accommodate the heterogeneity of the RL pool, the outcome verifier dispatches between two task-conditioned judge prompts. For closed-form analytic tasks with a gold answer, the verifier evaluates $\mathcal{V}_{\text{final}}$ against the reference along the axes of accuracy, completeness, and hallucination (Listing 6). For open-ended research tasks for which no gold answer exists, the verifier instead scores the relevance, analytical depth, and logical soundness of the submitted claims against the user request, conditioning additionally on a compact summary of the rollout evidence so that the score reflects whether the claims are supported by what the agent actually observed rather than by external priors of the judge (Listing 7). In both cases the judge is required to emit a structured JSON object with an ordinal score $s = \texttt{Judge}(q, \mathcal{G}^*, a^*) \in \{0, 1, 2, 3\}$ accompanied by a brief justification; with maximum rubric score $S = 3$ this yields $R_{\text{outcome}} = s/S \in [0, 1]$, and a malformed or unparseable judge response is treated as $s = 0$.

Listing 6: Outcome verifier system prompt for closed-form QA tasks.

```
You are an expert data analysis evaluator. Your task is to evaluate the quality of a set of
     analytical findings (claims) generated by an AI agent in response to a specific question. You
     will be provided with the user's question, the ground truth (golden answer), and the agent's
     generated claims.

Please evaluate the agent's claims based on how well they align with the golden answer.

### Evaluation Criteria:
1. Accuracy: Do the claims state facts that are correct according to the golden answer?
2. Completeness: Did the agent capture the core message and key points of the golden answer?
3. Hallucination: Did the agent invent data points, trends, or facts that contradict or are
     unsupported by the golden answer?

### Scoring Rubric (0-3):
* [3] Excellent (Fully Accurate & Complete): The claims perfectly capture the key points of the
     golden answer. There are no factual errors, no severe omissions, and zero hallucinations.
* [2] Partial (Minor Errors or Omissions): The claims capture the core of the golden answer but miss
     some minor details, or include slight, harmless extra information. No severe hallucinations or
     direct contradictions.
* [1] Poor (Severe Errors or Hallucinations): The claims contain significant factual errors,
     contradict the golden answer, miss the primary core message, or suffer from major hallucinations.

* [0] Invalid: The response is entirely empty, nonsensical, or completely unrelated to the question.

### Input Data:
[Question]: {question}
[Golden Answer]: {golden_answer}
[Agent Output]: {agent_claims}

### Output Format:
You must output your evaluation in the following JSON format. Provide a concise step-by-step
     reasoning BEFORE giving the final integer score.
{"reasoning": "Briefly compare the claims against the golden answer based on accuracy, completeness,
     and hallucination.", "score": <int between 0 and 3>}
```

Listing 7: Outcome verifier system prompt for open-ended research tasks.

```
You are a Senior Data Scientist and an expert evaluator. Your task is to evaluate the quality of
     exploratory data analysis findings (claims) generated by an AI agent for an open-ended research
     topic. There is no predefined correct answer.
```

```
Please evaluate the agent's claims based on their analytical value, logical soundness, and relevance
     to the research topic.

### Evaluation Criteria:
1. Relevance: Do the claims directly address the core intent of the research question?
2. Depth & Insightfulness: Do the claims demonstrate deep analytical thinking (e.g., correlations,
     anomalies), or are they just superficial observations?
3. Logical Soundness: Are the conclusions drawn plausible and logically supported by standard data
     analysis practices?

### Scoring Rubric (0-3):
* [3] Excellent (Highly Insightful & Logical): Highly relevant and logically sound. The claims go
     beyond obvious, surface-level observations to provide valuable, deep analytical perspectives.
* [2] Fair (Relevant but Superficial): The claims are relevant and generally logical, but lack
     analytical depth. They mostly state obvious facts or simple summaries without extracting deeper
     insights.
* [1] Poor (Flawed or Irrelevant): The claims draw highly questionable or illogical conclusions, lack
      structural coherence, or fail to meaningfully address the core research topic.
* [0] Invalid: The response is entirely empty, nonsensical, or completely ignores the research topic.

The user prompt additionally appends a compressed multi-turn evidence summary
(at most 6 messages, 280 chars each) so that the judge can verify whether the
final claims are supported by what actually happened during the rollout.

### Output Format:
{"reasoning": "Briefly evaluate the claims focusing on relevance, analytical depth, and logical
     soundness.", "score": <int between 0 and 3>}
```

Listing 8: Infer verifier system prompt, applied once per `infer` edge.

```
You are an expert evaluator for natural language inference in a claim-based data analysis workflow.

Your task is to evaluate whether a single infer step is logically valid based on its premises.

You will be given:
1. The original user question
2. A set of premise claims
3. The inferred conclusion
4. The reasoning text

### Scoring Rules
* [1] Good inference
  - The conclusion is supported by the premises.
  - The reasoning is logically sound and faithful to the premises.
  - No unsupported leap, hallucination, contradiction, or speculative extension is introduced.
* [-1] Bad inference
  - The conclusion is not supported by the premises, or overstates what the premises justify.
  - The reasoning introduces unsupported assumptions, speculative causal interpretation, hidden leaps,
      contradiction, or distortion.

### Important Instructions
- Judge ONLY whether this infer step itself is valid from the premises.
- Do NOT judge based on whether the final answer is correct.
- If the conclusion adds new causal/mechanistic/speculative interpretation not grounded in the
     premises, assign -1.
- Be conservative: assign 1 only when the inference is clearly justified by the premises.

### Output Format
{"reasoning": "Briefly explain whether the conclusion is supported by the premises and whether there
     is any unsupported leap.", "score": 1 or -1}
```

Reward-Model Overhead. The composite reward adds verifier traffic on top of standard outcome-only RL only through R_{infer} (one verifier query per `infer` edge) and R_{outcome} (one judge query per rollout); R_{process} is computed locally from execution feedback at no extra cost. Per task tuple this incurs $\mathcal{O}\big(N \cdot (|\mathcal{I}| + 1)\big)$ verifier calls, where $|\mathcal{I}|$ is empirically bounded by the number of derivational steps in a rollout (typically ≤ 10). We batch verifier calls across rollouts and use a small dedicated verifier rather than the policy itself, so the verifier is not on the critical path of the policy update; end-to-end this keeps reward-model overhead within roughly $1.3\times$ the rollout cost of an outcome-only baseline at the same N.

C Experimental Protocol

C.1 Benchmark Details

Datasets. We evaluate VeriGraph on four data-intensive benchmarks that collectively span single-table QA, multi-table data analysis, and multi-step research across heterogeneous sources. For all benchmarks, we utilize the official released splits without any further re-annotation. Specific details for each benchmark are delineated below:

- **TableBench** [47] (*Table QA*): TableBench is a comprehensive single-table question-answering benchmark encompassing four distinct reasoning skills: fact checking, numerical reasoning, data analysis, and visualization. We utilize the public test split and evaluate the $\sim$700 textual questions from the fact-checking, numerical-reasoning, and data-analysis subsets; the visualization subset is excluded, as it requires plot-based grading that is orthogonal to our focus on claim-grounding evaluation. Each instance comprises a single CSV or Markdown table paired with a natural-language question, where the expected answer is a short string, a numerical value, or a list.
- **InfiAgent-DABench** [14] (*Data Analysis*): InfiAgent-DABench targets closed-form data analysis on individual CSV files. We employ the official evaluation split consisting of 257 questions; each question specifies an analysis task—such as descriptive statistics, filtering-then-aggregation, or group-wise comparison—alongside an output format constraint. The gold answers are concrete values or short tuples; models are required to load the CSV, execute the analysis, and generate an answer that strictly adheres to the specified format.
- **DSBench** [23] (*Multi-table/Long-context Data Analysis*): DSBench evaluates agents on realistic data-science workflows involving multiple related tables and supplementary documents. We focus on the data-analysis track and evaluate 466 tasks; each task provides a bundle of CSVs—often containing hundreds of columns and tens of thousands of rows—plus a task description. The goal is to provide a numerical or short-text answer that necessitates joining, filtering, and aggregating across multiple tables. Given that the inputs frequently exceed standard context windows, DSBench specifically tests an agent's capability to navigate large workspaces through code execution.
- **DAB-Step Research** (*Multi-step Research*): DAB-Step [8] is a multi-step benchmark requiring joint reasoning over structured tables and unstructured policy or documentation files within a payments-processing context. While the original benchmark focused on simple QA, we evaluate on the 100-case subset released by DeepAnalyze [51], adopting their official split for direct comparability. Each instance includes a question demanding cross-source reasoning, the associated tabular inputs, and relevant documentation snippets. The outputs are open-ended research-style responses rather than short answers; following [51], an LLM judge scores each response on *Content* (factual correctness and completeness relative to the gold reference) and *Format* (structure, citation discipline, and presentation) on a 0–5 scale, and we report the average scores.

Grounding Rate (GR). On all four benchmarks we additionally report the Grounding Rate defined in Appendix C.3. GR is computed from the model's own output and evidence context and therefore does not depend on dataset-specific gold annotations, which makes it directly comparable across the four benchmarks above.

C.2 Baseline Details

We group baselines into three categories. All baselines receive the same user question and the same attached files. For QA-style benchmarks we extract the content inside `<answer>...</answer>` when present; for report-style tasks we pass the full generated report to the judge.

(i) General-purpose LLMs, direct prompting. The direct-inference baselines in Table 1 are GPT-5.2, Gemini-2.5-Pro, Claude-4.5-Sonnet, Claude-4.5-Opus, Qwen3-32B, and Qwen3-30B-A3B. They do not receive a Python tool. The evaluator renders each attached file into text before prompting: text-like files are read directly, spreadsheet / parquet / pickle files are converted through pandas previews, and the input is truncated to at most 20K characters per file and 60K characters in total. The exact system and user prompt template is shown in Listing 9.

Listing 9: Direct-inference baseline prompt template.

```
System:
You are a rigorous data analysis assistant.

You will receive a user question plus the textual contents of any attached files.
Files may be tables, HTML pages, JSON, Markdown, logs, source code, or other text-like data.

Your task:
1. Use only the provided question and file contents.
2. Inspect the file contents carefully before answering.
3. If the attached content is tabular, reason from the rows, columns, headers, and values shown.
4. If the attached content is HTML, reason from the visible text and relevant structured markup.
5. If the evidence is incomplete or truncated, say what can be answered from the visible content and
     avoid inventing missing facts.
6. Provide the final answer inside <answer></answer> tags. For report-style requests, put the full
     report inside those tags.

User:
Question:
{question}

Attached file contents:
{rendered_file_block}

Answer the question using the provided content. Put the final answer inside <answer></answer> tags.
```

(ii) ReAct / CodeAct data agents. The ReAct-style baselines are GPT-5.2, GPT-5.4, Claude-4.5-Sonnet, Claude-4.5-Opus, Qwen3-8B, Qwen3-Coder-30B, QwQ-32B, Qwen3-32B, and Qwen3-30B-A3B. Each model is wrapped by the same CodeAct scaffold: it can issue Python code in `<code_interpreter>` blocks, receives executor output in a tool-response block, and must finish with `<answer>...</answer>`. The maximum interaction budget is 30 Python calls, with a 1,200-character cap on each tool observation. The prompt in Listing 10 is the system prompt used by the scaffold; the implementation includes an additional one-shot height/weight correlation example after these instructions.

Listing 10: ReAct / CodeAct baseline system prompt.

```
You are a rigorous Data Analysis Agent operating within a persistent Python environment. Your core
     objective is to solve user's question through iterative code execution and produce a final
     answer.

## Iterative Workflow
1. Analyze: Write your internal reasoning process inside <think> tags.
2. Act: Execute Python code using the <code_interpreter> format.
3. Observe: You will receive the output in a <tool_response> block. Repeat until you have sufficient
     data.
4. Answer: Provide your final answer inside <answer></answer> tags. Whether the answer is a specific
     small number or a report, enclose it with <answer> and </answer> tags.

Before conducting the analysis, you should first check the data and the question to understand the
     user's intent.

## Tool Usage Format
To execute code, strictly use the following tag structure.
<code_interpreter>
# Write valid Python code here
import pandas as pd
print("Hello World")
</code_interpreter>

## Python Environment Rules
- Persistence: Variables persist across turns (Jupyter-style). You can directly use the variables
     defined in the previous turns.
- Output: Use print() to display results you need to see. Due to the limitation on the length of your
      context, consider the length of printed content and only print what is necessary.
- Files: If a file path is provided, it is in the current directory. Do not write new files.
- Do not write anything into local files.

## Answer Format
If you have sufficient information to solve the user's problem, terminate the tool invocation and
     wrap your answer with <answer> and </answer>.
- For short-answer questions, directly provide the short answer inside.
```

```
- For report-generation questions, wrap the report content inside <answer> and </answer>. Do not
    write your report in code.
```

Prompt-Veri. Prompt-Veri uses Qwen3-32B in the VeriGraph runtime with the same graph schema, code tools, and `bind/infer` interface, but without graph-specific SFT or RL. It tests whether prompting alone induces reliable evidence construction. Its backbone is larger than the trained 8B policy, so the comparison is not a same-backbone training ablation. Appendix D.9 reports both 30-turn and 50-turn settings. Document-citation prompting links text to source spans, whereas this interface also expresses runtime-artifact attachments and claim-to-claim dependencies.

(iii) Specialized data agents. For DataMind [38] and DeepAnalyze [51], we use the official model checkpoints and public inference scaffolds, keeping their native prompting and code-execution conventions. We only adapt file paths and answer extraction to match our benchmark harness. DeepAnalyze is run with a 30-round code-execution budget; generated `<Code>` blocks are executed and returned as `<Execute>` observations until the model emits `<Answer>` or the round budget is exhausted.

C.3 Grounding Rate Evaluation

To quantify the second evaluation axis used in Section 4.1, we define the **Grounding Rate (GR)** as claim-level support recoverability from the evidence artifact exposed by a method.

For each sample x_i, we first take the model's final output text a_i and decompose it into a set of atomic factual claims using an independent LLM call:

$$\mathcal{C}_i = \{c_{i1}, c_{i2}, \ldots, c_{i|\mathcal{C}_i|}\}.$$

The decomposition prompt requires the LLM to split the answer into minimal standalone factual claims, preserve the original meaning, avoid introducing new facts, and remove purely rhetorical or hedging text. If the answer is empty or contains no factual content, the claim set is empty. Each atomic claim is therefore treated as a minimal, semantically self-contained factual statement, and the decomposition process does not introduce information beyond the final answer.

For each atomic claim c_{ij}, we then construct the evidence artifact exposed by the method:

- **CodeAct**: the complete solving trajectory is used as the evidence context, i.e., `thought + code + observation`;
- **VeriGraph**: the final evidence subgraph $\mathcal{G}^*$ is used as the evidence context, concretely realized as the set of claim nodes reachable from the terminal claim node along ancestor relations.

This matches the user-facing audit interface of each method: a linear agent exposes a trajectory, whereas VeriGraph exposes a terminal evidence subgraph.

Given the evidence context, we represent its contents as a set of evidence units. To retrieve candidate evidence for each claim, we score evidence units using three complementary heuristics:

1. numeric overlap;
2. lexical token overlap;
3. exact substring match.

For numeric overlap, we first extract numeric expressions from both the atomic claim and each evidence unit using a separate LLM-based numeric extractor. The extractor is constrained to return only numeric expressions explicitly appearing in the input, including integers, decimals, percentages, currency amounts, ratios, year-like numbers, and signed numbers, without inventing additional values.

For each claim c_{ij}, we keep the top-k evidence units with the highest heuristic scores, denoted as its candidate evidence set $\mathcal{E}_{ij}$. The same segmentation, retrieval, and top-k selection procedure is applied to all methods before judging. An independent LLM judge then determines whether $\mathcal{E}_{ij}$ is sufficient to support the claim c_{ij}. The judge is given the user question, the atomic claim, and the

candidate evidence units, and returns binary labels indicating whether the evidence is relevant to the claim and whether it sufficiently supports the claim. We use the support label to define:

$$s(c_{ij}) \in \{0, 1\},$$

where $s(c_{ij}) = 1$ only if the candidate evidence sufficiently supports the claim, and $s(c_{ij}) = 0$ otherwise. The judge is instructed to be conservative: evidence that is missing, unrelated, contradictory, or too weak is treated as unsupported.

The **Grounding Rate (GR)** is finally defined as:

$$\mathrm{GR} = \frac{\sum_i \sum_{j=1}^{|\mathcal{C}_i|} s(c_{ij})}{\sum_i |\mathcal{C}_i|}.$$

GR measures the proportion of atomic claims in the model's final answer that can be explicitly supported by the evidence artifact the method produces. Because the denominator contains only emitted claims, we report GR alongside task scores that measure correctness and completeness. This evaluation separates factual claim decomposition, candidate evidence retrieval, and final support judgment, reducing the chance that unsupported claims are counted as grounded merely due to surface-level overlap. The exact prompts used for atomic claim decomposition, numeric expression extraction, and evidence judging are provided in Section C.5.

C.4 Evaluation Prompts

The judge prompt for QA accuracy, the rubric prompt for research-task scoring, and the prompt for our *Grounding Rate* judge are listed in Listings 11–12.

Listing 11: QA accuracy judge prompt template.

```
Compare the predicted answer against the golden answer.
Return whether the prediction is correct, allowing equivalent numeric formats and units.
```

Listing 12: Grounding Rate judge prompt template.

```
Given an atomic claim and a candidate evidence set drawn from the method's own evidence context,
     decide whether the evidence is sufficient to support the claim. Return a binary support decision.
```

For RL training, the outcome judge receives the task query, task metadata, the golden answer or official rubric, the terminal answer emitted by the agent, and the serialized terminal evidence subgraph. The judge assigns an ordinal score $s \in \{0, 1, \ldots, S\}$, where S is the maximum score of the corresponding task rubric. Exact-answer QA tasks use their binary or normalized accuracy rubric, whereas open-ended research tasks use the official content / format rubric inherited from the benchmark.

The rubric checks three aspects in order:

1. **Answer correctness**: whether the final answer matches the golden answer or satisfies the task-specific content rubric.
2. **Completeness**: whether all requested quantities, comparisons, and caveats are covered.
3. **Faithfulness to evidence**: whether the terminal answer is supported by the submitted evidence graph rather than by unsupported statements outside the graph.

The normalized reward used in Eq. 11 is s/S. If terminal extraction fails or no terminal claim is submitted, the outcome reward is set to zero.

C.5 Grounding Rate Evaluation Prompts

We use three LLM prompts in the Grounding Rate evaluation pipeline: one for decomposing final answers into atomic factual claims, one for extracting numeric expressions used in numeric-overlap retrieval, and one for judging whether the retrieved candidate evidence supports each atomic claim.

Listing 13: Atomic claim decomposition prompt.

```
You decompose an answer into atomic factual claims.

Return strictly valid JSON with this schema:

{
  "claims": ["claim 1", "claim 2"]
}

Rules:
- Split the answer into minimal standalone factual claims.
- Keep the original meaning.
- Do not add new facts.
- Remove purely rhetorical or hedging text.
- If the answer is already one atomic claim, return one item.
- If the answer is empty or contains no factual content, return {"claims": []}.
- Output JSON only.
```

Listing 14: Numeric expression extraction prompt.

```
You extract numeric expressions from text.

Return strictly valid JSON with this schema:

{
  "numbers": [
    {
      "text": "original numeric expression as it appears in the input"
    }
  ]
}

Rules:
- Extract only numeric expressions that are part of the answer/claim content.
- Include integers, decimals, percentages, currency amounts, ratios, year-like numbers, and signed
    numbers.
- Do not invent numbers.
- Preserve the surface form from the input text as closely as possible.
- If there are no numeric expressions, return {"numbers": []}.
- Output JSON only.
```

Listing 15: Grounding Rate evidence judge prompt.

```
You are a rigorous evidence judge following an ALCE-style claim support protocol.

You will receive:
1. A user question
2. One atomic claim
3. Candidate evidence units from the method-specific context

Your task is to decide:
- supported: whether the provided evidence units contain enough evidence to support the claim
- relevant: whether the provided evidence units are genuinely relevant to the claim rather than
    mostly unrelated

Return strictly valid JSON:

{
  "supported": 0 or 1,
  "relevant": 0 or 1,
  "reasoning": "brief explanation"
}

Judging rules:
- supported = 1 only if the evidence shown is sufficient for the claim.
- relevant = 1 if the evidence shown is directly about the claim, even if it is not fully sufficient.
- If the evidence is missing, unrelated, contradictory, or too weak, supported = 0.
- Be conservative.
- Output JSON only.
```

D Additional Analysis

D.1 Cost and Latency Analysis

Figure 5 reports the average token cost and wall-clock latency of representative Qwen3-based direct and tool-using systems. For tool-using systems, token counts are separated into reasoning tokens and code/action tokens; direct-prompting baselines are shown as single-stream answer generation. The latency measurements include Python-execution overhead for tool-using systems and evidence-graph materialisation for VeriGraph. As expected, VeriGraph incurs additional runtime on the long-evidence benchmarks where it builds larger terminal graphs, while the inference-time cost remains a single rollout without extra verifier calls.

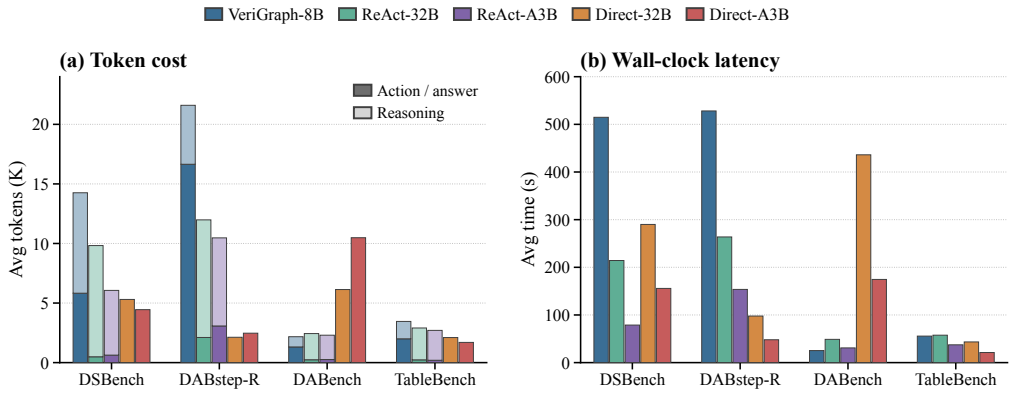


Figure 5: Token cost and wall-clock latency across representative Qwen3-based baselines. **(a)** Average generated tokens, where tool-using systems are split into reasoning and code/action tokens and direct systems use a single answer-generation stream. **(b)** Average wall-clock time per example.

Beyond the overall cost comparison, Figure 5 also reveals a clear difference in how VeriGraph and ReAct allocate their token budgets. Compared with ReAct, VeriGraph tends to spend a larger fraction of its tokens on code/action generation, whereas ReAct consumes most of its tokens in the reasoning stream. This suggests that VeriGraph effectively shifts part of the reasoning burden from unconstrained natural-language deliberation to executable programs. In other words, operations that require repeated reflection and verification in ReAct can be externalized into code execution in VeriGraph, allowing intermediate computations and evidence aggregation to be performed more deterministically. This design explains why VeriGraph may introduce additional execution latency, but also supports more structured and verifiable reasoning. Overall, the results show a task-dependent accuracy–auditability–cost trade-off: evidence-producing rollouts can be substantially longer on DSBench and DAB-Step Research. Distillation and external reward verifiers are training-time components; inference uses one rollout without additional semantic-verifier calls. Graph maintenance uses local deterministic operations, but its isolated timing requires a separate profiling protocol.

D.2 Analysis of Context Management

Table 6 compares the baseline runtime with the context-compressed variant across four representative benchmarks. The main pattern is that context compression does not necessarily reduce generation volume: in several settings the code-token budget actually increases, because the policy can afford to emit a more explicit executable trace once earlier history has been compacted. Even so, the wall-clock time is lower on most benchmarks, which is consistent with the reduced effective sequence length seen by the model at each step. In other words, compression shifts computation away from repeatedly reprocessing a long transcript and toward a slightly longer code trace, and the latter is cheaper than carrying the full uncompressed context through every turn.

The table suggests a clear efficiency trade-off. Across DSBench, DABstep-R, and DABench, the compressed-context variant reduces runtime while maintaining broadly similar aggregate scores, and it does so even when the number of generated code tokens rises. This indicates that the dominant cost is not token emission itself, but the repeated processing of a long conversational history. Context compression lowers that overhead by shortening the sequence carried into later turns, so the model

Table 6: Context-management comparison between the baseline runtime and the context-compressed variant. Token counts and wall-clock time are rounded to whole numbers; aggregate scores are rounded to three decimals.

	VeriGraph				**VeriGraph-context**			
Dataset	Thought	Code	Time	Score	Thought	Code	Time	Score
DSBench	8431	5826	515	66.43	13953	5824	368	65.12
DABstep-R	4950	16649	528	3.31	5890	17328	413	3.02
DABench	858	1316	25	85.99	666	1291	20	84.05
TableBench	1460	1997	56	73.58	1603	2639	61	72.06

can spend more of its budget on problem-specific code generation rather than on re-reading prior text. The tablebench result is a mild exception on runtime, but the overall trend remains that context compression improves execution efficiency with only limited impact on the final score.

D.3 Failure Modes in RL Training

This subsection summarizes the main failure modes we observed when optimizing the VeriGraph policy with reinforcement learning. Since the reward design is specified in Appendix B.2, we focus here on practical sources of instability and on the supervision signals required to make RL reliable for evidence-graph construction.

Outcome-only RL is too coarse for graph construction. Optimizing only the terminal answer reward provides no explicit feedback on the quality of the intermediate evidence graph. In our experiments, this induces a length bias: when multiple correct trajectories appear in a group, longer rollouts receive more total token-level updates because the same trajectory-level advantage is assigned to every generated token. As a result, outcome-only RL can favor verbose and redundant derivations, even when a shorter trajectory is equally correct and yields a cleaner support graph. For graph construction, additional steps are not inherently beneficial; they enlarge the failure surface and increase the risk of unsupported claims or unnecessary detours.

Direct RL from the base model rarely reaches valid terminal states. Before the policy learns the atomic graph-writing format, many rollouts fail before they can be judged: the model may omit required primitives, violate the output schema, or terminate without producing a submittable final claim set. In this regime, the final judge is rarely invoked, making the reward extremely sparse and largely uninformative. We therefore use a cold-start stage with distilled graph-augmented trajectories, which gives the policy sufficient syntactic and operational competence to explore the intended RL objective rather than spending most updates on format violations.

Executability can mask weak semantic transitions. The process reward provides the first dense signal by checking whether executed actions are valid. This is crucial in early training, where malformed code is a dominant failure mode. However, executability alone does not ensure that a trajectory is useful for evidence construction. A rollout can be fully executable while still containing weak or unsupported semantic transitions. Edge-level supervision on `infer` is therefore essential: it distinguishes locally justified derivations from merely plausible ones and prevents process correctness from degenerating into graph-level noise.

Reliable training requires layered supervision. In practice, the strongest signal comes from combining process, `infer`, and outcome rewards. The process reward keeps the policy within the executable action space, the inference reward improves the quality of the support graph, and the outcome reward anchors optimization to end-task correctness. Our experience suggests that these signals should be used hierarchically rather than interchangeably: process supervision makes RL trainable, inference supervision makes the graph meaningful, and outcome supervision keeps the policy aligned with the final objective.

D.4 Cross-Model Consistency Analysis

We compare independent judge backends on final-answer correctness, local derivational validity, claim-level grounding support, and report scoring. Table 7 expands the initial 50-pair-per-view study to approximately 300 paired examples per view. Agreement is 93.6% for final-answer correctness, but lower for infer steps (81.0%) and claim support (82.0%). These latter views require semantic judgments; the observed disagreement motivates reporting human audits and complementary rule-based checks. Cross-judge agreement is an evaluator diagnostic and does not itself prove that either judge is correct.

Table 7: Expanded paired-judge comparison. Agreement and MAE are label-based for the categorical views. Research scores are compared by correlation; categorical agreement and MAE are not reported for this view.

View	Judge pair	N	Agree.	κ	MAE	Pearson	Spearman
Final answer	5.4-mini / QwQ	282	.936	.824	.064	.828	.828
Infer step	5.4-mini / 4o-mini	300	.810	.589	.380	.603	.603
Claim support	5.4-mini / QwQ	300	.820	.573	.180	.576	.576
Research score	5.4-mini / QwQ	290	—	—	—	.775	.753

Here 5.4-mini, 4o-mini, and QwQ denote `gpt-5.4-mini`, `gpt-4o-mini`, and QwQ-32B. The research score combines Content and Format on the 0–5 scale.

Figure 6 retains the original 50-pair-per-view pilot for comparison. It uses score-valued GR and GPT-5.4 versus QwQ-32B for the research panel, whereas Table 7 reports the expanded claim-support and judge-backend study. The sample sets and evaluation views must not be pooled.

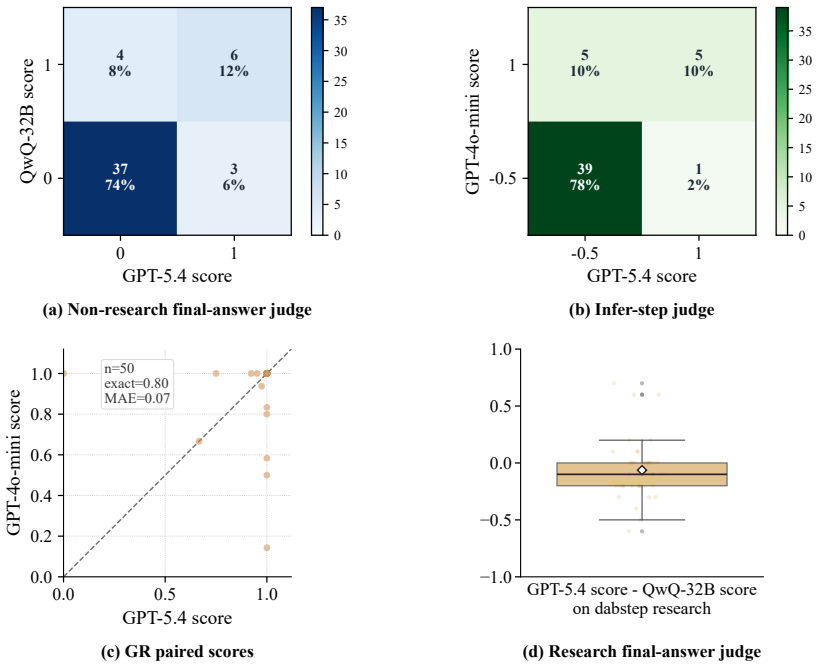


Figure 6: Original cross-model pilot, with 50 sampled pairs in each view. **(a)** Final-answer confusion matrix. **(b)** Infer-step confusion matrix. **(c)** Paired GR scores, with exact agreement and MAE. **(d)** Signed normalized research-score differences between GPT-5.4 and QwQ-32B; Content and Format are averaged and normalized to $[0, 1]$. Expanded results are in Table 7.

D.5 Independent Human Audit

Two non-author annotators assessed 180 outputs containing 605 claims from TableBench and DABStep Research. The audit asks whether each claim is supported by the evidence artifact exposed by the method, rather than whether the final answer merely matches a gold answer. Table 8 provides the complete breakdown. Human GR pools supported claims, following Appendix C.3; the aggregate values are 60.45% for VeriGraph, 42.57% for Claude CodeAct, and 19.47% for Qwen CodeAct. The displayed method names follow the audit summary rather than asserting an undocumented model version.

Table 8: Human audit on two datasets. Human and LLM GR and agreement are percentages on this audit subset. κ measures human–LLM agreement, not agreement between the two human annotators. Time is median seconds per output.

Dataset	Method	Claims	Human GR	LLM GR	Agree.	κ	Time (s)
DABStep-R	Claude CodeAct	126	17.46	28.57	77.78	.384	150.0
	Qwen CodeAct	150	8.67	4.00	91.33	.276	59.5
	VeriGraph	134	50.75	68.66	73.13	.460	115.0
TableBench	Claude CodeAct	76	84.21	81.58	97.37	.907	75.0
	Qwen CodeAct	76	40.79	61.84	55.26	.143	48.0
	VeriGraph	43	90.70	81.40	90.70	.620	40.0

The human scoring preserves VeriGraph's lead, but absolute scores change substantially on research claims. Relative to Claude CodeAct, the median audit time is lower for VeriGraph in both datasets; these observational timings do not establish a causal reduction in analyst effort. The methods emit different numbers of claims, so GR must be read alongside answer completeness. This audit is limited to two datasets and does not calibrate the absolute GR of every benchmark or baseline.

D.6 Rule-Based Evaluation Checks

We re-score answer correctness with question-type-specific regular-expression extraction, normalization, and exact matching against gold answers (Table 9). This scoring does not use an LLM judge. It is less tolerant of semantically equivalent phrasing than holistic judging, and its scores are lower for all listed configurations. The ordering of the three methods is preserved on both datasets.

Table 9: Rule-based answer checks. Accuracy and agreement are percentages; n is the reported number of evaluated outputs. Coverage differs on TableBench, so these are diagnostic comparisons rather than a paired common-sample test.

Dataset	Method	n	Rule acc.	LLM acc.	Agree.	κ
TableBench	Qwen3-32B Direct	836	65.91	69.98	81.82	.583
	Qwen3-32B CodeAct	836	66.39	67.34	85.17	.665
	VeriGraph-8B	825	72.36	73.58	84.73	.613
DABench	Qwen3-32B Direct	257	17.90	22.57	81.32	.423
	Qwen3-32B CodeAct	257	76.65	84.82	82.49	.443
	VeriGraph-8B	257	81.32	85.99	84.44	.433

For grounding, Sim-GR checks whether an output sentence is recoverable from an earlier trajectory sentence with similarity greater than 0.95. Num-GR checks whether output numbers occur in the prior trajectory or can be recomputed from executed code. Both scoring rules avoid a semantic LLM judge. They measure surface recovery and numerical recoverability, respectively, and do not establish whether an artifact supports the meaning of a claim. Their higher absolute scores should therefore not be read as semantic GR. Table 10 reports all available counts and scores; the scoring stage is distinguished from any upstream claim decomposition.

D.7 Bootstrap Uncertainty in Overall Performance

Table 11 reports the 95% intervals from the supplementary bootstrap analysis using 10,000 resamples. The overlapping intervals motivate describing task performance as competitive rather than claiming

Table 10: Complementary rule-based grounding checks. Scores are percentages; evaluated output and claim counts differ across methods.

Dataset	Method	Outputs	Claims	Sim-GR	Num-GR
TableBench	Qwen3-32B CodeAct	279	969	89.16	97.16
	VeriGraph-8B	792	2909	89.76	99.92
DABench	Qwen3-32B CodeAct	155	350	91.38	97.76
	VeriGraph-8B	251	1079	94.29	98.83

statistically significant superiority over proprietary agents. Overlap does not prove equivalence, and these Overall intervals do not quantify uncertainty in GR or repeated-training variation.

Table 11: Reported 95% bootstrap intervals for Overall performance, with 10,000 resamples. Point scores remain those in Table 1.

Method	95% interval
VeriGraph-8B	[71.46, 75.86]
Claude-4.5-Opus ReAct	[71.16, 75.25]
GPT-5.4 ReAct	[69.39, 74.54]

D.8 Sensitivity to Evidence Retrieval

To examine whether top-k retrieval alone explains the grounding gap, the judge is given all available evidence units on matched 50-case samples per dataset for Qwen3-32B CodeAct and VeriGraph. Table 12 compares this full-evidence condition with the default retrieval condition on the same samples. Removing retrieval improves CodeAct's three-dataset average from .464 to .523, while VeriGraph changes from .877 to .889. The remaining gap shows that retrieval budget does not fully explain the difference in these subsets; this is not a benchmark-wide recalculation of GR.

Table 12: GR sensitivity on matched 50-case subsets. Scores are proportions; Avg. is the mean across the three datasets.

Dataset	Method	Default top-k	Full evidence
DABench	CodeAct	.816	.841
	VeriGraph	.838	.853
DSBench	CodeAct	.116	.095
	VeriGraph	.814	.826
TableBench	CodeAct	.460	.632
	VeriGraph	.980	.986
Avg.	CodeAct	.464	.523
	VeriGraph	.877	.889

D.9 Matched Interaction-Budget Baselines

The main baseline protocol uses at most 30 Python calls, whereas VeriGraph uses up to 50 interaction turns. To test this difference, CodeAct and Prompt-Veri are rerun at 50 turns with the same observation caps, file access, generation settings, and answer-extraction protocol as VeriGraph. Table 13 reports both settings. CodeAct improves on DSBench and TableBench, but does not close the gap to VeriGraph. Prompt-Veri approaches VeriGraph on DABench, while CodeAct remains stronger on DAB-Step Research. The schema without task-specific training is therefore insufficient for the observed gains across all tasks.

D.10 Comparison with Tree-Search Inference

A search tree represents alternative execution states and explores them to select a trajectory. The evidence DAG records dependencies within an execution: artifacts support atomic claims, and

Table 13: Interaction-budget comparison. CodeAct and Prompt-Veri use Qwen3-32B. The first three rows are accuracy proportions; DABStep-R is the mean Content/Format score on a 0–5 scale.

Dataset	CodeAct 30	CodeAct 50	Prompt-Veri 30	Prompt-Veri 50	VeriGraph-8B
TableBench	.6734	.6970	.6490	.6440	.7358
DABench	.8482	.8500	.8440	.8590	.8599
DSBench	.5644	.6058	.5129	.5598	.6643
DABStep-R	3.7650	3.7900	2.4600	2.9300	3.4400

premise claims support derived conclusions. Retaining intermediate text in a search state does not by itself encode these typed relations. The two structures can be combined by maintaining an evidence graph inside each search state.

Table 14 reports an MCTS-ReAct comparison on 50 examples from each of three datasets. Search improves the task score over single-rollout CodeAct, but remains below VeriGraph on these samples with reported relative inference costs of 3.36–$4.16\times$. These subset results do not establish that all search methods or budgets behave similarly, and do not replace a comparison with search applied on top of VeriGraph.

Table 14: Tree-search comparison on 50 examples per dataset. Scores are accuracy proportions; the cost column reproduces the reported relative inference-cost multipliers.

Dataset	CodeAct	MCTS-ReAct	VeriGraph	Reported cost ($\times$)
DSBench	.540	.570	.660	3.36
DABench	.820	.840	.900	4.16
TableBench	.640	.690	.760	3.91

E Case Studies

To concretely illustrate how VeriGraph operationalizes claim-level verifiability across qualitatively different analytic regimes, we present two complementary case studies that bracket the spectrum of tasks targeted by our framework. The first (Appendix E.1) is a *compact, closed-form* decision-support task in which a single small table admits a short deterministic derivation and a single recommended action; it is intended to expose, in fully traceable form, the elementary edge types (computation, `bind`, `infer`) that constitute an evidence graph. The second (Appendix E.2) is an *open-ended, multi-source* research-report task drawn from the DABstep-research, in which the agent must synthesize dozens of statistics computed over heterogeneous files into a multi-paragraph narrative; it is intended to demonstrate that claim-level provenance remains tractable, and uniformly auditable, as the answer surface and the underlying reasoning chain grow by an order of magnitude. Together, the two cases show that the same graph abstraction governs both extremes, yielding answers whose every numeric assertion can be traced back along a single edge to the computation node that produced it.

E.1 Decision Support over a Tabular Source

We revisit, in fully expanded form, the warehouse-restocking instance previewed in Figure 4(c). The user issues a decision-support query asking which of three warehouses should be prioritized for restocking, given a single tabular source listing each warehouse's current inventory, daily demand, and replenishment lead time (Table 15). Despite its modest size, the instance exercises the full VeriGraph pipeline end to end: deterministic computation over raw cells, `bind` operations that lift numeric results into verifiable claims, and a terminal `infer` step that composes those claims into the recommended action.

The agent first computes three executable quantities for each warehouse. The *days-to-stockout* value measures how many days current inventory can support demand:

$$\text{days}_i = \frac{\text{inventory}_i}{\text{daily_demand}_i}.$$

Table 15: Raw table used in the warehouse-restocking case study.

Warehouse	Inventory	Daily demand	Lead time
A	120	30	6
B	200	20	4
C	90	15	3

The *replenishment gap* compares this quantity with lead time:

$$\text{gap}_i = \text{days}_i - \text{lead_time}_i.$$

Finally, the non-negative shortage-risk score is

$$\text{risk}_i = \max(0, -\text{gap}_i).$$

This gives $\text{days}_A = 4$, $\text{days}_B = 10$, and $\text{days}_C = 6$; $\text{gap}_A = -2$, $\text{gap}_B = 6$, and $\text{gap}_C = 3$; and risk scores $\text{risk}_A = 2$, $\text{risk}_B = 0$, and $\text{risk}_C = 0$. Warehouse A is the only warehouse whose inventory is expected to run out before replenishment arrives.

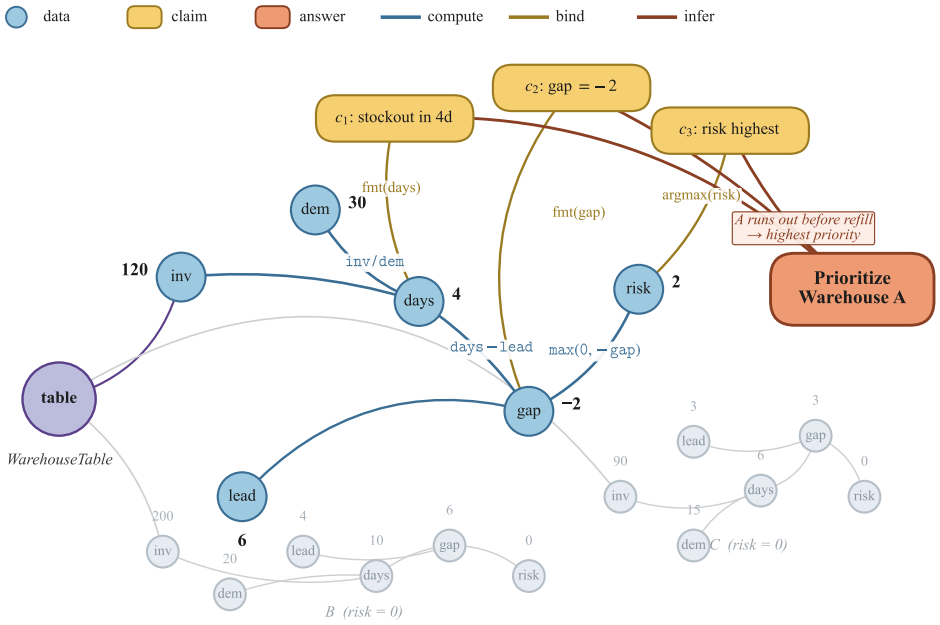


Figure 7: Detailed evidence graph for the warehouse-restocking case. The highlighted path is the selected answer chain for Warehouse A; faded nodes show the alternative computations for Warehouses B and C. For readability, the figure expands bound runtime values into explicit data nodes, while the implemented runtime exports the corresponding claim-level DAG.

The evidence graph in Figure 7 separates the answer into three layers. First, data-to-data computation edges derive `A_days`, `A_gap`, and `A_risk` from the raw table fields. Second, `bind` edges convert selected computed artifacts into natural-language claims:

- c_1: Warehouse A will stock out in 4 days.
- c_2: Warehouse A has a replenishment gap of -2 days.
- c_3: Warehouse A has the highest restocking risk score, with risk 2 versus 0 for both alternatives.

Third, a single `infer` step combines c_1, c_2, and c_3 into the final claim c_4: Warehouse A should be prioritized for restocking. The faded B/C branches are not part of the final answer path, but they

are important audit evidence: they show that the agent did not choose A merely because A has low inventory, but because the computed risk comparison makes A the only positive-risk warehouse.

Listing 16: Abbreviated VeriGraph trajectory for the warehouse-restocking case.

```
<think>
I need to identify the warehouse with the highest stockout risk. I will compute
days-to-stockout, replenishment gap, and a non-negative shortage-risk score for
each warehouse, then bind the decisive facts and infer the final recommendation.
</think>
<code_interpreter>
warehouse_table = {
    "A": {"inventory": 120, "daily_demand": 30, "lead_time": 6},
    "B": {"inventory": 200, "daily_demand": 20, "lead_time": 4},
    "C": {"inventory": 90, "daily_demand": 15, "lead_time": 3},
}

days = {
    k: v["inventory"] / v["daily_demand"]
    for k, v in warehouse_table.items()
}
gap = {
    k: days[k] - v["lead_time"]
    for k, v in warehouse_table.items()
}
risk = {k: max(0, -gap[k]) for k in warehouse_table}
priority = max(risk, key=risk.get)

c1 = bind(
    "Warehouse A will stock out in {a_days:.0f} days.",
    a_days=days["A"],
)
c2 = bind(
    "Warehouse A has a replenishment gap of {a_gap:.0f} days.",
    a_gap=gap["A"],
)
c3 = bind(
    "Warehouse A has the highest restocking risk score: A={a_risk:.0f}, B={b_risk:.0f}, C={c_risk:.0f
     }.",
    a_risk=risk["A"], b_risk=risk["B"], c_risk=risk["C"],
)
</code_interpreter>
<tool_response>
Execution Status: Success
Current claims:
[c1] c1 [Context]: Warehouse A will stock out in 4 days.
[c2] c2 [Context]: Warehouse A has a replenishment gap of -2 days.
[c3] c3 [Context]: Warehouse A has the highest restocking risk score: A=2, B=0, C=0.
</tool_response>
<think>
The decisive facts are now claims: A runs out before replenishment and has the
only positive risk score. I can submit the derived recommendation.
</think>
<code_interpreter>
c4 = infer(
    premises=[c1, c2, c3],
    conclusion="Warehouse A should be prioritized for restocking.",
    reasoning=(
        "Warehouse A has only 4 days of inventory against a 6-day lead time, "
        "creating a -2 day replenishment gap, and its risk score of 2 exceeds "
        "the zero risk scores for Warehouses B and C."
    ),
)
submit_answer(c4)
</code_interpreter>
```

This example illustrates the local auditability targeted by VeriGraph. If the final answer is challenged, a reviewer can inspect the specific edge where the concern arises: the raw table extraction, the deterministic computation of `days`, `gap`, or `risk`, the `bind` operation that turns a computed value into a claim, or the `infer` operation that combines the three claims into the final recommendation.

E.2 Multi-Source Research Report on a Payments Dataset

Whereas Appendix E.1 examined a closed-form decision over a single tabular source, we now turn to a substantially more demanding regime that more faithfully reflects practical analyst workloads. The instance is drawn from the open-ended split of DABstep and is characterized by an intentionally under-specified prompt, namely "Create a research report summarizing key observations or notable points in the data," paired with seven heterogeneous source artifacts: two textual schema descriptions (`payments-readme.md`, `manual.md`), a fact table of 138,236 payment transactions recorded in 2023 (`payments.csv`), a merchant directory (`merchant_data.json`), a fee-rule catalogue (`fees.json`), and two reference dimensions (`merchant_category_codes.csv` and `acquirer_countries.csv`). In contrast to the warehouse case, the expected deliverable is an open-ended, multi-paragraph narrative rather than a single scalar answer, and the space of admissible claims is correspondingly large. This case therefore probes whether claim-level provenance scales gracefully when the answer surface itself is generative.

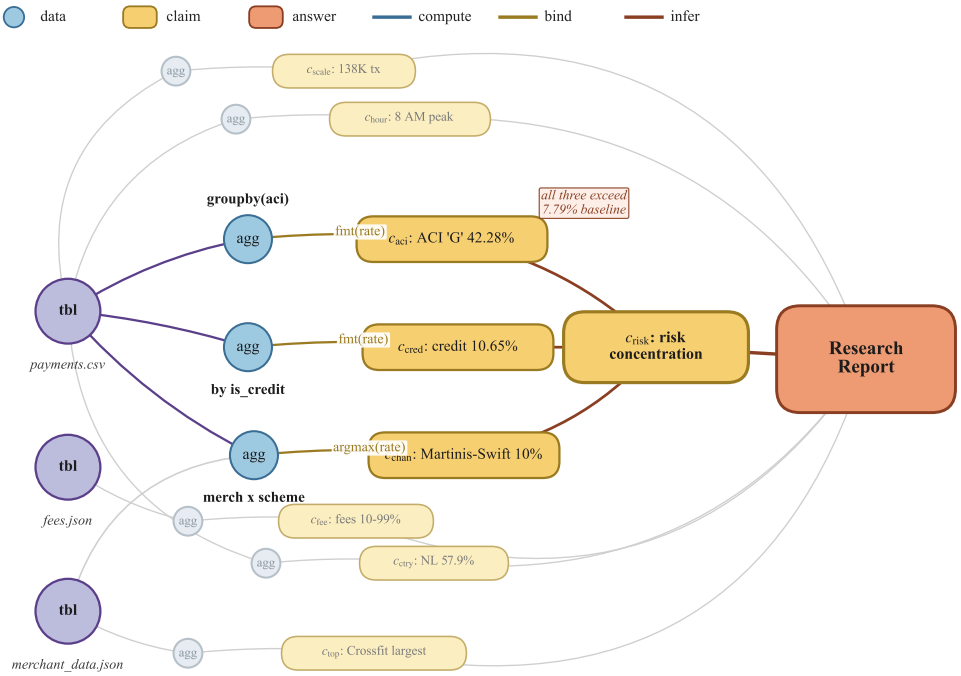


Figure 8: Evidence graph for the payments research-report case. Three source tables (left) feed deterministic group-by computations, and each computation is lifted into a verifiable claim by a `bind` edge. The highlighted hero subgraph shows a fan-in `infer` step that derives an intermediate claim c_{risk} (drawn with a thicker border to mark it as derived rather than directly bound) from three independent binds (ACI breakdown, credit versus non-credit, and merchant$\times$card-scheme), all of which exceed the 7.79% dataset-wide dispute baseline. Faded satellite claims (e.g., hourly peak, country share, and fee distribution) feed the final `Research Report` answer node directly. Every sentence in the report can be traced backwards along a single bind edge to the computation node that produced its numeric value.

This setting exposes two failure modes that prose-only agents commonly suffer from. First, with dozens of statistics flowing into a single document, it is easy for the writer to introduce a number that no upstream computation actually produced, or to silently re-use a stale value after a later recomputation, and the reader has no efficient way to localize either error. Second, the agent must coordinate across multiple files and group-bys (by merchant, card scheme, hour, ACI code, country, and device), and the chain of pandas operations that justifies any single sentence is typically long.

What the evidence graph contains. Over 29 reasoning turns, the policy first issues schema-discovery code, then deterministic aggregation code that produces one statistic per group-by, and finally a sequence of `bind` edges that lift each computed scalar into a natural-language claim. The resulting graph contains 33 final claims, an abbreviated list of which is reproduced in Listing 18. Because every claim is anchored to the exact computation node that produced its numeric value, the report is no longer a wall of statistics but rather a view over a checkable graph. Table 16 shows representative claim families together with the upstream computations they bind to.

Trajectory shape. Listing 17 shows an abbreviated slice of the trajectory. A deterministic group-by produces the per-ACI dispute rate, a single `bind` step lifts the salient row into a verifiable claim, and a subsequent fan-in `infer` step composes a "risk-concentration" claim from the ACI, credit-card, and merchant-channel binds. The same pattern is repeated for each claim family, and the final report node is then an `infer` over the selected claim subset.

Listing 17: Abbreviated VeriGraph trajectory for the payments research-report case.

```
<think>
The request is open-ended. I will materialise one statistic per group-by,
bind each into a claim, and only then synthesise the report so that every
sentence is traceable to a specific computation.
</think>
<code_interpreter>
import pandas as pd, json
payments = pd.read_csv("payments.csv")
fees     = pd.DataFrame(json.load(open("fees.json")))
merchants = pd.DataFrame(json.load(open("merchant_data.json")))

# --- per-ACI dispute rate ---
aci_stats = (
    payments.groupby("aci")["has_fraudulent_dispute"]
            .agg(["mean", "count"])
            .sort_values("mean", ascending=False)
)
top_aci          = aci_stats.index[0]
top_aci_rate     = aci_stats.loc[top_aci, "mean"]
top_aci_count    = int(aci_stats.loc[top_aci, "count"])
other_aci_rate   = aci_stats.drop(top_aci)["mean"].max()

c_aci = bind(
    "ACI code '{aci}' has a dispute rate of {rate:.2%} over {n} transactions, "
    "compared to a max of {other:.2%} for the remaining ACI codes.",
    aci=top_aci, rate=top_aci_rate, n=top_aci_count, other=other_aci_rate,
)
</code_interpreter>
<tool_response>
Execution Status: Success
Current claims:
[c_aci] ACI code 'G' has a dispute rate of 42.28% over 25463 transactions,
        compared to a max of 0.00% for the remaining ACI codes.
</tool_response>
<think>
I now repeat the same pattern for the credit-card flag, the
merchant$\times$card-scheme cross-tab, the hourly profile, the fee
catalogue, and the per-country split. Each statistic becomes its own bind.
Once all binds are in place, a single infer step composes the report.
</think>
<code_interpreter>
# ... per-merchant, per-hour, per-card-scheme, fees, country binds ...

c_risk = infer(
    premises=[c_aci, c_credit, c_merchant_channel],
    conclusion=(
        "Fraud risk is concentrated in specific channels: ACI 'G' (42.28%), "
        "credit cards (10.65%), and Martinis_Fine_Steakhouse-SwiftCharge (10%)."
    ),
    reasoning=(
        "All three binds independently identify channels whose dispute rate "
        "exceeds the 7.79% dataset-wide baseline by a large margin."
    ),
)

c_report = infer(
    premises=[c_scale, c_overall, c_aci, c_merchant_top, c_hour,
              c_country, c_fee, c_credit, c_risk, ...],
```

```
    conclusion="<multi-paragraph report bound to the above claims>",
)
submit_answer(c_report)
</code_interpreter>
```

Table 16: Representative claim families in the payments research report and the deterministic computations they are bound to. Each row corresponds to a distinct subgraph in the evidence graph, and the final report is an `infer` step over a selected subset of these claims.

Claim family	Upstream computation	Example bound claim
Dataset scale	`len(payments)`, `nunique(merchant)`, `nunique(card_scheme)`	138,236 transactions across 5 merchants and 4 card schemes.
Overall fraud rate	`mean(has_fraudulent_dispute)`	Overall fraudulent-dispute rate is 7.79%.
ACI breakdown	`groupby(aci).agg(rate, count)`	ACI code `G`: 42.28% dispute rate over 25,463 transactions, vs. 0% for other ACI codes.
Per-merchant ranking	`groupby(merchant).agg(...)`	`Crossfit_Hanna` is the largest merchant (55,139 transactions, mean 92.07 EUR).
Hourly fraud profile	`groupby(hour_of_day).mean(...)`	Hour 8 has the highest dispute rate (6.43%) within the 8–10 AM window.
Channel risk concentration	join over `aci`, `is_credit`, `merchant`×`card_scheme`	Risk is concentrated in ACI `G` (42.28%), credit cards (10.65%), and `Martinis_Fine_Steakhouse-SwiftCharge` (10.00%).
Fee structure	`describe(rate, fixed_amount)` on `fees.json`	Fee rates span 10%–99% (mean 54.3%); fixed amounts 0–0.14 EUR (mean 0.069 EUR).
Geographic concentration	`groupby(acquirer_country).size()`	Netherlands accounts for 57.9% of acquired volume.

Auditability under realistic length. This case makes concrete why the evidence graph matters more, not less, as the answer becomes longer. The final report contains roughly twenty distinct numeric claims spread over several paragraphs. Without per-claim provenance, a reviewer who suspects, for instance, the 42.28% figure for ACI `G` would have to re-derive it from the raw trajectory log. With VeriGraph, the same reviewer follows a single edge from the sentence to its bind, and then from the bind to the deterministic `groupby(aci)` computation that produced it. The same property applies to fan-in claims such as the risk-concentration sentence: the reviewer can verify each premise independently and check that the `infer` step does not introduce an unsupported numeric value.

A second observation is that the graph cleanly exposes redundancy and mild inconsistency that prose alone would hide. Several binds in this case, such as two slightly different statements of the same hourly fraud peak, map to overlapping computation nodes, so that the deduplication needed to produce a clean report becomes a graph operation rather than a re-reading of free text. We view this as a representative example of the kinds of audits that become tractable once the answer is a graph.

Listing 18: Final claim set produced for the payments research-report case (abbreviated).

```
[c01]  138236 transactions in 2023, 5 merchants, 4 card schemes.
[c02]  Overall fraudulent-dispute rate: 7.79%.
[c03]  ACI 'G': 42.28% dispute rate over 25463 transactions; other ACI codes ~0%.
[c04]  Crossfit_Hanna is the largest merchant: 55139 transactions, mean 92.07 EUR.
[c05]  Per-merchant dispute rates are tightly clustered (7.68%--8.00%).
[c06]  Acquirer countries: 5 distinct; Netherlands dominates with 57.9%.
[c07]  Hour 8 is the peak transaction hour; highest hourly dispute rate in 8--10 AM.
[c08]  Card schemes: GlobalCard / NexPay / TransactPlus / SwiftCharge, 13733--48150 tx.
[c09]  Fee rates: min 10%, max 99%, mean 54.3%; fixed amounts 0.00--0.14 EUR.
[c10]  Device mix: 'Other' (22.4%) and Android most frequent.
[c11]  Credit-card dispute rate: 10.65% vs. 0.00% non-credit.
[c12]  49 distinct merchant category codes (3000--9399).
[c13]  Highest merchant-card-scheme dispute rate: Martinis_Fine_Steakhouse-SwiftCharge (10%).
[c14]  Lowest merchant-card-scheme dispute rate: Belles_cookbook_store-GlobalCard (7.11%).
[c15]  Total transaction value: 12,697,297.46 EUR; mean 91.85 EUR.
```

```
... (further binds for fee distribution, MCC ranges, and merchant directory metadata)
[c33]  Risk-concentration infer: ACI 'G', credit cards, and Martinis-SwiftCharge
       jointly account for the channels whose dispute rate exceeds the 7.79% baseline.
```

F Reproducibility and Compliance

F.1 Compute Resources

All training and evaluation runs were performed on a cluster of A100-80G GPUs.

- **SFT**: $4\times$ A100-80G GPUs, bfloat16, DeepSpeed ZeRO-3, approximate wall-clock time 24 hours.
- **RL**: $8\times$ A100-80G GPUs, with rollout generation and policy updates co-located on the same nodes; approximate wall-clock time 48 hours.
- **Inference / evaluation**: $4\times$ A100-80G GPUs for agent rollouts and LLM-as-judge evaluation. Average per-benchmark wall-clock time on VeriGraph-8B is approximately 1.5 hours (TableBench), 0.5 hours (InfiAgent-DABench), 1.0 hour (DSBench), and 0.8 hours (DAB-Step Research).

Including preliminary runs, ablations, and failed experiments not reported in the main text, the full research project consumed roughly $1.5\times$ the budget summarized above.

F.2 Licenses for Existing Assets

We use the following upstream assets in compliance with their respective licenses. All datasets are used solely for non-commercial research evaluation; we do not redistribute the upstream data or model weights.

- **TableBench** [47]: Apache License 2.0.
- **InfiAgent-DABench** [14]: Apache License 2.0.
- **DSBench** [23]: MIT License.
- **DABstep** [8]: Apache License 2.0.
- **Qwen3-8B** [45] (base model): Apache License 2.0.
- **MS-Swift** [57] (SFT framework): Apache License 2.0.
- **verl** [42] (RL framework): Apache License 2.0.

Our own code and trained checkpoints will be released under the Apache License 2.0, consistent with the licenses of the upstream training frameworks and base model.

G Limitations and Future Directions

While VeriGraph achieves strong empirical results, several limitations remain that also point to promising directions for future work.

Internalizing the evidence graph at scale. Constrained by compute budget and the size of the available trajectory data, our post-training is performed at a relatively small scale. Although this is already sufficient to deliver consistent gains, we believe that scaling up post-training, or injecting structured evidence-graph supervision earlier in the pre-training stage, would let the model internalize graph construction as a native reasoning skill rather than a learned interface, further improving both the quality and the efficiency of the resulting reasoning.

General task coverage. Our experiments and evaluation focus on data-intensive scenarios, where traceability and factuality are particularly important and currently under-served. As agent capabilities continue to mature and real-world deployments expand, we expect attention to shift from raw accuracy toward auditability and faithfulness across many more task families. Extending VeriGraph to multimodal reasoning, cross-document synthesis, and long-horizon tasks that span extended time frames would expose new structural challenges (e.g., visual evidence, retrieval provenance, temporal dependencies) and new opportunities for evidence-graph supervision.

Richer reinforcement-learning signals. The composite reward we introduce already yields meaningful improvements, but its current realisation is intentionally conservative: it builds on relatively simple graph-structural features and edge-level verifier checks. A natural next step is to evaluate the entire evidence graph with more expressive models, for example graph neural networks that score global coherence and informativeness, or learned critics calibrated against audited reasoning errors. Incorporating targeted human feedback into the reward design is another promising avenue, which could guide the policy toward constructing higher-quality, more genuinely useful evidence graphs.

H Statement on the Use of LLMs

During the preparation of this manuscript, we use LLMs as a general-purpose assistance tool. The primary role of the LLM is to aid in improving the clarity and readability of the text, as well as to accelerate the implementation of our research ideas. Specific applications include: (1) Language and Grammar Correction: Polishing sentence structure, correcting grammatical erros, and refining word choices to enhance the overall quality of the writing. (2) Paraphrasing and Style Refinement: Rephrasing sentences and paragraphs to ensure consistency in tone and style throughout the paper. (3) Code Implementation Assistance: Generating code snippets and providing debugging support to help implement the proposed algorithms and experimental setups.

It should be noted that all core research concepts, experimental design, data analysis, and conclusions are developed exclusively by the human authors. Any content or suggestions generated by the LLM, including code, are critically checked, and substantially edited by the authors to ensure accuracy. The authors take full responsibility for the final content of this paper.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the paper's main contributions: a heterogeneous evidence DAG with computational, grounding, and derivational primitives; graph-based policy optimization; and empirical validation of accuracy and grounding. These claims are supported by the methodology in Sections 3–3.4, the main results in Table 1, and the traceability and ablation analyses in Sections 4.4 and 4.5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss limitations in Appendix G, including incomplete low-level provenance, semantic-judge disagreement, task-dependent rollout cost, limited human-audit coverage, and the evaluated 4B–14B backbone range.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not include formal theorems or proofs; the contribution is an empirical neuro-symbolic framework with an associated training and evaluation protocol.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 4.1 specifies the benchmarks, baselines, backbone (Qwen3-8B), interaction-turn and length budgets, SFT setup (MS-Swift, 36K instances, learning rate $1e{-5}$), and RL setup (Verl with DAPO, 8 rollouts per prompt, learning rate $1e{-6}$). Appendices A.1, B.2, and C describe the runtime, training protocol, prompts, and evaluation procedure, and the anonymized repository linked in the abstract provides the implementation path for reproduction.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The abstract links an anonymized repository for VeriGraph. The release is intended to include the training and evaluation code, prompts, and the curated DAB-Step Research subset, together with instructions to reproduce the main results.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4.1 reports the evaluation benchmarks and sizes, baseline families, backbone, context window, maximum generation length, maximum interaction turns, SFT/RL frameworks, learning rates, rollout count, and training-data size; Appendices B.1, B.2, and C provide additional data, training, and evaluation details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Appendix D.7 reports 95% bootstrap intervals from 10,000 resamples for three Overall comparisons. We retain a conservative No because the available experiment summary does not specify the resampling unit and aggregation protocol completely, and no paired significance test or repeated-training uncertainty is reported.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix F.1 reports the compute budget for each stage: SFT used $4\times$ A100-80G GPUs for approximately 24 hours; RL used $8\times$ A100-80G GPUs (rollout generation co-located with policy updates) for approximately 48 hours; inference and LLM-as-judge evaluation used $4\times$ A100-80G GPUs, with average per-benchmark runtime reported alongside each baseline. Preliminary and ablation experiments consumed roughly an additional $1.5\times$ the reported training budget.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

Justification: The work uses publicly available benchmarks and models, and we have reviewed the NeurIPS Code of Ethics. The added human audit uses two non-author annotators to assess evidence support; participant documentation and the applicable institutional review status require author confirmation.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: Appendix G discusses positive impacts and potential negative impacts (overtrust in seemingly verifiable yet imperfect reasoning, and the risk that auditable analytics tooling is misapplied to sensitive data).

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The released artifacts (training/evaluation code and a curated subset of an existing public benchmark) are derived from publicly available data-analysis benchmarks and pose no high misuse risk.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring

that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.

- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All upstream assets are credited and used in accordance with their licenses, which are summarized in Appendix F.2. Specifically: **TableBench** (Apache-2.0), **InfiAgent-DABench** (Apache-2.0), **DSBench** (MIT), and **DABstep** (Apache-2.0) are used as research benchmarks under their respective open-source terms; **Qwen3-8B** is used under the Apache-2.0 license; the training framework **MS-Swift** (Apache-2.0) and the RL framework **verl** (Apache-2.0) are used as released. We do not redistribute the upstream datasets or model weights; only derived evaluation outputs and our own code (released under Apache-2.0) are included with the submission.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The work introduces release artifacts including the VeriGraph implementation, prompts, and the curated DAB-Step Research subset. The abstract links the anonymized repository, and the paper describes the subset, training/evaluation setup, and evaluation protocol in Section 4.1 and Appendices B.1, B.2, and C.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [No]

    Justification: Appendix D.5 reports an audit by two non-author annotators and the measured audit times. The available materials do not include the full participant instructions, compensation details, or adjudication procedure, so these are not claimed to be fully disclosed.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [No]

    Justification: The added human audit involves two non-author annotators. The available materials do not document the applicable institutional review or exemption, consent process, or participant-risk disclosure; no approval or exemption is claimed here.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are a core component of VeriGraph: Qwen3-8B serves as the agent backbone trained via SFT and RL, several proprietary and open-source LLMs are used as baselines, and LLM-as-judge is used for rewards and evaluation. These roles are described in Section 4.1 and Appendices C.2, and G.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.