Understanding Reliability Bottlenecks in LLM Agents for HPC I/O Telemetry Analysis

Chris Egersdoerfer*, Orcun Yildiz†, Kevin Harms†, Robert Latham†, Robert Ross†, Dong Dai*
* University of Delaware, Newark, Delaware, USA
Email: cegersdo@udel.edu, dai@udel.edu
† Argonne National Laboratory, Lemont, Illinois, USA
Email: {oyildiz,harms,robl,rross}@anl.gov

***Abstract*—High-performance computing (HPC) facilities collect extensive I/O telemetry, but turning that data into operator-relevant insight remains a manual, expert-driven process. Tool-using LLM agents offer a promising interface for automating this workflow, yet it remains unclear how reliable they are on realistic telemetry-analysis tasks and which design choices materially improve that reliability. To answer this question, we conduct a controlled measurement study using `CIODI-Bench`, a set of 49 expert-curated tasks built from more than two years of public production telemetry from the Polaris supercomputer. We use `CIODI-Bench` as an evaluation basis, enabling controlled comparisons of tool-using agents for HPC I/O telemetry analysis. Our central finding is that the dominant reliability bottleneck shifts with task structure: on constrained extraction tasks, curated semantic grounding raises *average-of-3* performance from 73.0% to 93.5% (a 20.5-point gain); on data-analysis tasks, additional process-aware review steps improve the *average-of-3* performance by 13.2 points over the corresponding baseline, whereas on insight-generation tasks the same added LLM agent orchestration does not improve performance. However, the more elaborate orchestration increases output cost by almost 3×. These results clarify the reliability limits of current agentic approaches in this domain and provide practical guidance for building more trustworthy telemetry-analysis agents.**

I. Introduction

High-performance computing (HPC) storage systems underpin large-scale data-intensive workloads and are typically operated as shared facilities where many jobs contend for common storage resources [1], [2], [3], [4]. To operate and monitor these systems effectively, facilities collect extensive I/O telemetry, including job-level profiler traces, filesystem monitoring data, and scheduler or maintenance logs [5], [6], [7], [8], [9], [10], [11], [12]. Despite this mature data collection infrastructure, turning raw logs into operator-relevant answers remains largely manual and expert-driven because it typically requires bespoke multi-source queries, careful validation of domain-specific semantics, and repeated iteration [13].

Agentic LLM systems that can interact with databases, code execution environments, and external APIs [14], [15], [16], [17] offer a compelling path toward automating this workflow. In principle, an analyst could ask a natural-language question, have an agent retrieve the relevant evidence, run the necessary computations, and return an auditable answer. This vision has motivated a growing body of work on LLM agents for end-to-end data analytics in general domains [18], [19]. Yet for HPC facilities, the central question is not whether an agent can produce a plausible answer once, but whether it can do so reliably on realistic telemetry-analysis tasks, and which design choices materially change that reliability.

Answering this question requires evaluation on tasks that reflect the structure of real telemetry analysis. Unfortunately, existing benchmarks are not a good fit. Text-to-SQL and database-grounded QA benchmarks such as BIRD, Spider 2.0 and BEAVER primarily center on query generation and execution accuracy, missing out on significant downstream processing and analysis workflows [17], [15], [20], [21]. Recent data-science and data-analysis agent benchmarks broaden the workflow by using more general code execution environments and posing longer analytical tasks, yet they typically target general-purpose analytical settings rather than domain-based reasoning over operational telemetry and profiler data [22], [23], [24], [25], [26], [27]. As a result, they still do not reveal how agents behave when success depends on multi-source joins, domain semantics tied to the underlying schema, open-ended interpretation, and abstention when the available data schema is insufficient.

We therefore conduct a controlled measurement study of tool-using LLM agents for HPC I/O telemetry analysis. Specifically, we examine how expert-defined semantic context, agent orchestration methodology, and model capability affect accuracy, repeated-run reliability, and output cost across tasks with different structures and difficulty levels. To hold the data, tools, and scoring procedure fixed across these comparisons, we construct and use `CIODI-Bench` (Complex I/O Data Insights Benchmark, pronounced 'coyote-bench'), a benchmark of 49 expert-curated tasks over more than two years of public production telemetry from the Polaris supercomputer at Argonne National Laboratory [28].

As a result, we find that both agent capability and reliability are shaped less by any single design choice than by the structure of the task itself. Concretely, the dominant bottleneck shifts from *semantic grounding* on constrained extraction tasks, to *verification tractability* on semi-constrained analysis tasks, to *upstream hypothesis formation* and *solvability judgment* on open-ended tasks. This bottleneck migration means that the same intervention (e.g., curated context, process-aware review, or a stronger model) can be decisive for one task regime yet nearly irrelevant for another, while heavier orchestration can increase cost by more than 8× without comparable

SC26, November 15-20, 2026, Chicago, Illinois, USA
979-8-3195-4789-7/26/$31.00 ©2026 IEEE

gains. We also observe that, on the hardest tasks, failures often stem from missing task-specific semantic disambiguation rather than missing tool access alone, suggesting that expert-in-the-loop mechanisms remain an important systems problem for trustworthy deployment. For the HPC storage community, these results mean the path to trustworthy agent-assisted telemetry analysis is not simply to adopt stronger frontier models or more elaborate multi-agent pipelines. It is to identify which parts of the workflow can be reliably automated, which require vetted semantic grounding, and where expert oversight must remain part of the loop.

Together, this work yields three contributions. First, we present a systematic measurement study of how agent context, orchestration, and model capability affect accuracy, stability, and cost in realistic HPC telemetry analysis. Second, we show that reliability bottlenecks shift systematically across task regimes, yielding concrete guidance for HPC facilities on when vetted semantics, lightweight verification, stronger models, or explicit expert escalation are most valuable. Third, we introduce `CIODI-Bench` as a publicly available evaluation substrate: 49 expert-curated tasks over production Polaris data, designed to stress multi-source joins, domain-specific semantics, open-ended analysis, and reliable abstention.

The remainder of this paper is organized as follows. Section II reviews the challenges of delivering insight from telemetry logs and presents three motivating cases from which we derive concrete benchmark requirements. Section III describes the design of `CIODI-Bench`, including task construction, output contracts, and per-run scoring. Section IV describes the evaluated agent configurations and measurement protocol. Section V presents experimental results on accuracy, stability, cost, and failure modes. Section VI discusses related work, and Section VII concludes.

II. Background and Motivation

We provide the necessary background and discuss three examples of applying LLM agents to I/O telemetry analysis that motivate the design of `CIODI-Bench`.

A. From Telemetry to Decisions

Modern HPC facilities generate abundant I/O telemetry, including job-level profiles such as Darshan [5] and Recorder [10] logs, scheduler/accounting records, and operational status logs [6], [7], [8], [29], [30]. These sources often share stable join keys, making relational databases a practical foundation for large-scale telemetry analysis through a common SQL-and-scripting workflow. What remains far less mature is reliably converting these signals into trustworthy insights as operator-relevant answers rarely reduce to a single query or data source. Rather, they require synthesizing evidence across sources and making a sequence of domain-specific judgment calls.

Consider a user asking "Why did my job have $3\times$ slower I/O than last week?" Answering this seemingly simple question typically requires joining profiler records across multiple runs, checking whether the job hit a different filesystem or allocation, and comparing I/O patterns (e.g., access sizes, shared vs. independent access, or metadata intensity) [31]. Even when the relevant data can be retrieved, correct interpretation depends on domain semantics, such as which Darshan counters correspond to throughput-relevant time, what differences in node allocations matter, and whether the observed change exceeds normal run-to-run variability.

While there is a rich ecosystem of dashboards, canned queries, and workflow scaffolding intended to reduce friction, their impact is often limited as the core analytical burden still falls on human experts [32], [33], [34]. Dashboards and observability platforms (e.g., Grafana-style views over time-series telemetry) make known metrics easy to monitor and drill into, and facilities often pair these dashboards with notebook-driven workflows for deeper investigations [35], [36]. Yet these tools are primarily effective when analyses are repetitive and match existing views or vetted query templates. When questions are novel or diverse, experts must spend significant effort to write database queries, implement downstream scripts, and validate each intermediate step.

This distinction defines the target of our study. For recurring, well-understood analyses, dashboards, curated queries, and templates remain preferable because their procedures can be validated once and reused. We instead focus on the long tail of evolving, multi-source questions for which the relevant data, joins, analytical choices, and validation steps are not known in advance. In this setting, end-to-end autonomous agents represent a useful measurement target as evaluating what they can accomplish independently reveals both the potential scope of automation and the points at which domain experts must remain involved.

B. Agents as a Potential Bridge

Recent advancements in large language models (LLMs) have enabled tool-using agents that can interact with databases and code-execution environments [16], [37], [38], [39], making them a plausible interface for automating knowledge work and data analysis across domains [40], [41], [42], [43], [44], [18], [45], [46], [47], [48], [19], [49]. For HPC telemetry, the appeal is immediate. Instead of writing SQL and analysis scripts by hand, an analyst could ask, "Compare throughput for jobs using MPI-IO collective writes versus independent writes," and expect the agent to retrieve the relevant data, perform the computation, and return an auditable answer.

However, real-world telemetry analysis is more nuanced than this example suggests. Prior work on data-analysis agents and decomposition-based workflows indicates that success often depends on more than base model capability, including the context available to the agent and how reasoning and verification are organized [18], [45], [19], [47]. Sparse schema context can induce semantic errors, while richer curated context may prevent them; similarly, explicit planning and verification pipelines may improve reliability, but they can also add cost without clear benefit. Whether these patterns hold in HPC telemetry analysis, and which kinds of tasks make them matter most, remains largely unknown. Answering such a question requires evaluation on realistic telemetry-analysis

tasks, as well as an understanding of what kinds of successes, instabilities, and failures such an evaluation must capture. We therefore turn to three representative examples that ground the design requirements of CIODI-Bench.

C. Motivating Examples

The three examples below are chosen to span reliable success, unstable performance, and consistent failure. Together, they show why evaluation in this domain cannot stop at whether an agent produces a plausible answer once; it must also account for defensible analytical variation, run-to-run stability, and tasks that expose current capability limits.

Where agents succeed

What is the monthly node-hours percentage consumed by jobs with Darshan logs?

Agents solve this by aggregating node-hours from scheduler records and joining on Darshan-profiled job IDs. They usually complete this workflow reliably, although valid choices such as assigning a job to the month of its start time or end time can produce slightly different aggregates.

Where agents are unstable

Is MPI-IO usage more prevalent for larger jobs?

A natural approach is to bin jobs by node count and compare MPI-IO prevalence across bins. But node counts are highly skewed, so naive bins can create misleading trends by collapsing very different scales (e.g., [1–2], [3–6], [7–1024]).

Where agents fail

For jobs with high MPI-IO volume during shared-file access, what is the estimated number of aggregators?

Darshan does not report aggregator counts directly, but in some cases, an expert can estimate them by comparing per-file MPIIO volume with the slowest rank's POSIX I/O volume on the same file. The estimates are especially credible when they turn out to be feasible integers, such as 16 or 48 across multiple files in the same job. Agents consistently fail to identify such an approach, highlighting a current capability boundary and the continued need for expert oversight.

These examples motivate three requirements for evaluation in I/O telemetry analysis. First, because even correct analyses may differ in defensible ways, as the first example shows, scoring should be tolerance-aware and able to award partial credit rather than treating all deviations as equally wrong. Second, because the second example shows that small analytical choices can lead to unstable conclusions, evaluation should measure performance across repeated runs rather than relying on single-run accuracy alone. Third, because the final example exposes a real capability boundary, the task set should span a broad range of difficulty, including tasks that probe the limits of current agents. To make the resulting measurements interpretable, tasks should also be categorized along these dimensions so that differences across agent designs can be tied to specific strengths and weaknesses rather than averaged away. The next section turns these requirements into the design of CIODI-Bench, which we use not simply as a benchmark, but as the evaluation basis for a controlled study of agent reliability in realistic HPC I/O telemetry analysis.

III. CIODI-Bench

CIODI-Bench is designed around four properties that directly follow from these requirements: it operates over real production Polaris telemetry, includes administrator-facing analysis tasks as well as requests that are genuinely out of scope for the available data, pairs each task with an explicit output contract and expert-produced reference artifact for automatic scoring with partial credit, and organizes tasks so performance can be analyzed by structure and difficulty while holding the tool environment fixed. This allows differences across agents to be attributed to context, orchestration, and model choice rather than to changes in interface affordances.

A. Telemetry Data Substrate

CIODI-Bench tasks operate over a PostgreSQL database that links application-level Darshan I/O profiles from production Polaris jobs with Polaris operational context (scheduler records and machine-status/maintenance logs) through shared identifiers such as job IDs. Specifically, we integrate three public sources summarized in Table I: (i) Polaris job scheduler records, (ii) Polaris machine status logs, and (iii) an anonymized collection of Darshan logs captured from production Polaris jobs [50]. The scheduler and machine-status datasets are provided as CSVs through the ALCF reports portal [28] and are loaded with minimal transformation.

Data source	Coverage (start)	Records
Job records	2022	>580,000 jobs
Status records	2022	>500 events
Darshan profiles	2024	>1,200,000 logs

TABLE I: Datasets underlying CIODI-Bench.

Darshan logs are parsed from their binary format into tabular form and ingested as (i) per-module counter tables for STDIO, POSIX, and MPI-IO and (ii) per-module log-level summary tables containing the output of Darshan's log-level summary tooling (e.g., job_stats). Jobs traced by Darshan that exhibit no instrumented I/O are retained in a dedicated summary table. In total, the schema comprises two operational-context tables (scheduler and machine status), six per-module Darshan tables (counters and summaries), and one no-I/O table. This organization preserves the structure of the original sources while supporting the multi-step SQL+Python workflows central to CIODI-Bench tasks.

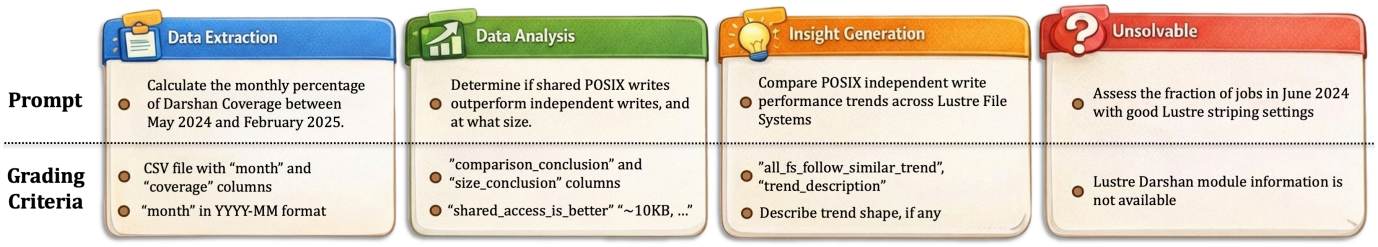


Fig. 1: Example tasks from each task type in CIODI-Bench.

B. Task Sourcing

To ensure CIODI-Bench tasks remain grounded in real administrator-facing analyses over production Polaris telemetry, we draw them from three sources: prior Polaris case studies, elicitation from I/O experts and national lab facility staff, and deliberately out-of-scope requests that test whether agents recognize when the available data is insufficient.

Seed tasks. We initialize the benchmark with tasks derived from a recent study of the Polaris logs [13]. That work reports a set of case-study analyses with corresponding ground-truth results, making it a natural seed for CIODI-Bench. We convert these artifacts into benchmark tasks by translating each analysis objective into a natural-language prompt and re-implementing the original scripts against our database to reproduce the reported ground truth. In total, seven CIODI-Bench tasks originate from this seed set.

Task elicitation from domain experts. To extend the task collection with additional realistic I/O telemetry analysis tasks, we solicited analytic task ideas from I/O experts and national lab facility staff via a survey form. The form asked respondents to imagine an autonomous agent with access to Polaris job scheduler records and Darshan I/O profiles, and to propose up to five analysis questions they would realistically ask in operational or investigative workflows. The prompt explicitly encouraged realistic tasks at a range of difficulty levels.

Unsolvable tasks. In addition to solvable tasks, we intentionally include tasks that are *out of scope* for the benchmark database. We collect these by explicitly asking the experts to propose analyses that would require missing signals (e.g., unavailable counters, absent system-level telemetry, or unmodeled per-rank timelines). These tasks are labeled as unsolvable and are used to evaluate whether agents can correctly abstain when a request cannot be answered from the available data. For example, one such task is "For jobs using Lustre file systems in 2024, what fraction had good striping settings in place?" Since the Polaris log collection does not include Lustre module data, this question cannot be answered verifiably.

C. Task Instances, Ground Truth, and Per-run Scoring

Each CIODI-Bench task instance consists of (i) a task prompt describing the analysis objective and (ii) a single reference CSV output file constituting the ground truth. As shown in Figure 1, each prompt includes a task definition describing what should be computed and grading criteria specifying the required CSV schema and representation conventions. This design keeps tasks open in how they are solved (e.g., through different SQL formulations and Python post-processing choices) while remaining precise about what must be produced.

The reference CSV serves as the authoritative target output for evaluation. Because tasks are phrased in terms of meaningful quantities or conclusions (e.g., node-hour totals, per-project aggregation, coverage rates), the reference artifact encodes the intended interpretation of the analysis objective. Per-run scoring compares the submitted CSV artifact against this reference. Building upon prior artifact-based evaluation (e.g., DA-Code's table comparison [51]), we use a normalization and column-wise grading pipeline that supports partial credit. Outputs are first normalized into a comparable representation by coercing column data types to expected types and standardizing common presentation variations, after which each column is scored according to its semantics (categorical labels, numeric quantities, short free-form text, etc.). The final task score is the mean of the per-column scores, yielding a scalar in $[0, 1]$.

For tasks with short natural-language explanations or summaries, enumerating all valid explanations as a small categorical label set would effectively reveal the set of hypotheses to test. For these cases, we use an LLM-as-a-judge method. To keep the judge lightweight and reproducible, it compares a single reference response against a single candidate response on a per-row basis, assigning 0.0 when the candidate misses all key information, 0.5 when it misses part of the key information, and 1.0 when it covers the required information. Each such column is paired with a task-specific prompt that specifies the key information the judge should focus on.

D. Task Taxonomy for Interpretable Analysis

To make reliability failures interpretable rather than collapsing all tasks into a single aggregate score, CIODI-Bench organizes tasks along two axes: **Task Type** and **Task Difficulty**. Table II defines the four task types used throughout the benchmark, while Figure 1 gives a representative instance of each. Table III reports the resulting distribution of tasks across type and difficulty. This taxonomy provides an interpretable breakdown of what the benchmark measures and supports the stratified analyses used later to isolate strengths and weaknesses across agent designs.

Task Type. Task types are distinguished primarily by grading modality and by the open-endedness of the required reasoning. *Data extraction* tasks are graded mainly through deterministic equality checks over numeric and categorical fields aggregated from the data. *Data analysis* tasks reduce to categorical labels

Type	Definition (what the agent must do and produce)
Data Extraction	Retrieve, join, aggregate, and format database fields into a structured CSV without interpretation of results.
Data Analysis	Retrieve and analyze evidence to emit a set of *categorical* conclusions (e.g., better/worse trend).
Insight Generation	Retrieve and analyze evidence to synthesize open ended conclusions and/or provide reasoning to explain findings.
Unsolvable	Reason about available data to accurately classify tasks as out of scope.

TABLE II: CIODI-Bench task types and their definitions.

produced by interpreting intermediate results. *Insight generation* tasks are more open-ended, requiring agents to synthesize conclusions or explain why observed findings support those conclusions. *Unsolvable* tasks are graded by whether the agent abstains and grounds that abstention in missing evidence.

Task Difficulty. We assign each solvable task a difficulty label (**Easy**, **Medium**, or **Hard**) using an operational, quantitative proxy: the number of distinct steps in the ground-truth solution (e.g., queries and post-processing operations) required to produce the final artifact. This criterion reduces subjectivity while reflecting a known challenge for tool-using agents, namely sustaining multi-step execution without drifting or accumulating semantic errors. Specifically, **Easy** tasks are close to directly observable in the data, requiring minimal aggregation and little higher-level reasoning. **Medium** tasks require mapping commonly used analysis concepts to the database schema (e.g., selecting the right counters, joins, and groupings). **Hard** tasks demand more challenging domain-specific reasoning and typically require longer multi-step workflows and substantial reasoning over intermediate results. Finally, **Unsolvable** tasks do not receive a difficulty label and are marked as **N/A** in Table III.

Tag	Easy	Medium	Hard	N/A	Total
Data Extraction	10	0	0	0	10
Data Analysis	0	14	2	0	16
Insight Generation	0	13	4	0	17
Unsolvable	0	0	0	6	6
Total	10	27	6	6	49

TABLE III: Task distribution by type and difficulty.

IV. Agent Configurations and Measurement

Using `CIODI-Bench`, we systematically study agent configurations that vary along two axes: curated *context* and explicit *orchestration*. To ensure fairness, all configurations operate in the same tool environment and solve the same tasks.

A. Shared Tool Environment

All agent configurations use the same three tools: a read-only SQL query tool, a persistent code execution tool, and an artifact submission tool. The query tool provides read-only access to the benchmark database. It returns a preview of up to 10 rows in the interaction loop while materializing the full result as a CSV in the agent's execution environment. The code execution tool allows agents to run Python or Bash code in a sandboxed Jupyter kernel that persists for the duration of the task, supporting iterative debugging and post-processing of query results. The artifact submission tool finalizes a run by submitting the path to the output artifact to be graded, after which the run terminates and the submitted artifact is passed to the grading pipeline (Section III-C). Holding this interface fixed ensures that performance differences are attributable to context, orchestration, and model choice rather than to differences in tool affordances.

B. Agent Context

Agents' behavior is strongly shaped by the information provided in their context, that is, by the tokens available in the message history. To isolate the impact of domain-specific context from other factors, we ensure every agent receives the same basic instructions covering how to call the three tools and how to operate in the environment. Beyond these shared instructions, the context includes two primary components.

Database knowledge documents the database schema, including table names and column meanings, and clarifies conventions that are easy to misuse in a purely schema-discovery workflow (e.g., whether a table is job-level or file-level, or whether values are aggregated across MPI ranks).

Domain knowledge encodes conventions that are not part of the database schema but are important for correct analysis. We derived these conventions from pilot experiments and domain-expert feedback, then presented them to agents as a small, dedicated part of the system prompt:

- *Filtering partial logs*: Agents treated partial logs as failures.
- *Misunderstanding rank semantics*: Agents misinterpreted rank values during processing, such as treating `rank=-1` incorrectly instead of recognizing that it denotes data shared across all ranks.
- *Misguided throughput calculation*: Agents used naive bytes/time calculations for throughput, ignoring more accurate methods based on actual I/O time.
- *Interface hierarchy*: Agents misunderstood how higher-level interfaces (MPIIO) are represented in lower-level interface (POSIX) records.

We distilled these corrections into general conventions rather than task-specific hints, improving robustness without leaking answers.

C. Agent Orchestration

Beyond context, performance also depends on orchestration design, which carries its own costs. It is therefore important to understand which orchestration strategies are actually effective. In this study, we organize representative strategies into an *orchestration ladder* with increasing structural complexity.

1. Direct-Solver. The Direct-Solver agent is a simple ReAct-style [41] loop that iteratively calls tools to (i) retrieve relevant data via SQL queries, (ii) perform any required post-processing in the code execution environment, and (iii) submit a single CSV artifact that satisfies the task's output contract.

This configuration serves as the minimal orchestration baseline because it introduces no explicit decomposition, verification, or additional LLM roles beyond the core tool-using loop.

2. Plan-and-Execute. Plan-and-execute separates planning from tool execution using a dedicated *planner* and *executor*, following Erdogan et al. [52]. The planner converts the prompt into an ordered sequence of steps, while the executor uses the available query and code-execution tools to complete each step and report a detailed summary of results. The goal of this approach is to reduce the task load assigned to any single role. A useful side effect is improved interpretability, since the planner's outputs make the high-level process explicit. We further evaluate two variants of this approach:

- *Upfront plan generation*. The planner produces a complete plan once; the executor executes it step by step and returns results for submission. On failure, the executor reports the issue and the planner revises the plan before continuing.
- *Iterative plan generation*. The planner emits one step at a time, conditioned on intermediate results returned by the executor. This is useful when the solution path is heavily dependent on intermediate results.

3. Reviewer Augmentation. We augment the plan-and-execute setting with a third LLM role, an external *reviewer*, to check intermediate assumptions and create a plan-execute-review loop. Specifically, after the planner and executor produce a candidate result artifact, we initialize a reviewer with (i) the original task prompt and output contract, (ii) the planner's proposed steps together with the executor's result summaries, and (iii) the final candidate CSV submitted by the original planner. The reviewer is instructed to validate the solution independently by verifying key decisions through appropriate cross-checks and by implementing alternative strategies that either confirm or call the existing result into question. The reviewer submits the final artifact upon completion.

In §V, we further distinguish reviewer configurations by the process information they receive, that is, whether the reviewer is *process-aware* and is shown the process that generated the original solution, or *process-blind* and must assess only the task and the final artifact.

D. Repeated-run Reliability Metrics

As outlined in Section II, a core requirement of this study is to measure not only whether an agent *can* solve a task, but whether it does so *reliably* under stochastic generation. To this end, we evaluate each task over K independent trials under identical settings and compute two standard metrics for every result. First, ***average-of-k*** is the mean score across the K trials and reflects expected performance under repeated execution. Second, ***best-of-k*** is the maximum score across the K trials and serves as an upper bound on attainable performance under favorable sampling. When summarizing results over multiple tasks, we aggregate these per-task quantities by averaging across tasks. Throughout the paper, we treat *average-of-k* as the primary reliability metric and report *best-of-k* as a complementary indicator of capability when reliability is not yet saturated.

V. Benchmarking Results and Analysis

This section presents the paper's core empirical results: a controlled measurement study based on `CIODI-Bench`. Rather than using the benchmark simply to rank agents, we use it to isolate which design choices materially change reliability, stability, and cost in HPC I/O telemetry analysis. We organize the results around three research questions (RQs):

- **RQ1:** How do expert-defined context and agent orchestration affect performance across `CIODI-Bench` task types (data extraction, data analysis, insight generation, and unsolvable)?
- **RQ2:** How do expert-defined context and agent orchestration affect performance across `CIODI-Bench` task-difficulty levels, and what failure modes dominate as difficulty increases?
- **RQ3:** What cost–performance tradeoffs arise from increasingly structured orchestrations, and when is additional structure worth the overhead?

A. Evaluation Setup

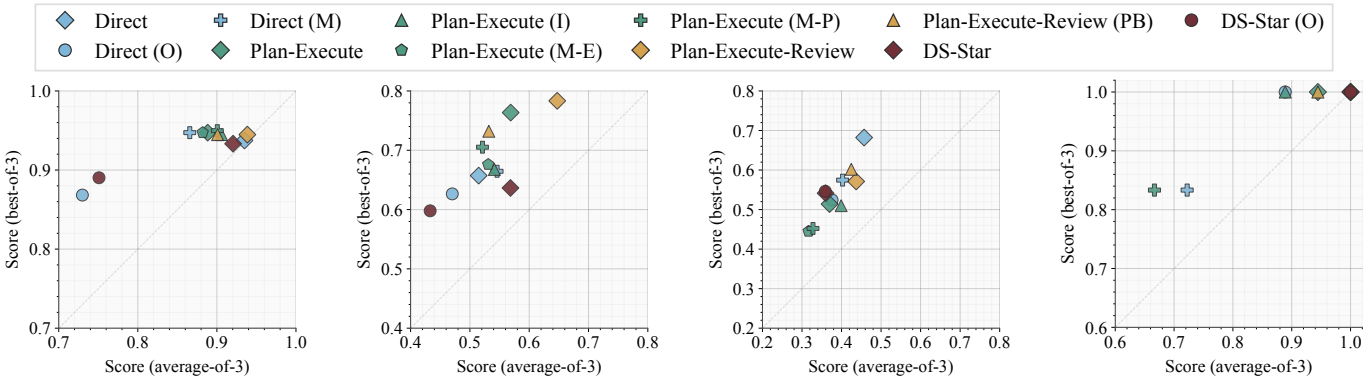
For each agent configuration, we follow the repeated-run protocol defined in Section IV-D with $K{=}3$ independent trials per task. We also standardize model inference settings across roles. Unless otherwise noted, all methods use OpenAI's GPT-5 as the underlying LLM. For the Direct-Solver and all planner roles in Plan-and-Execute setups, we set reasoning effort to *high*. For executor roles in Plan-and-Execute setups, we use the default setting (*medium*), reflecting a planner/executor division in which planning receives more deliberation while execution emphasizes tool use under the planner's instructions.

Benchmarked Variants. We report a *Direct-Solver* baseline in three variants. The plain variant (**`Direct`**) receives the detailed database-table context and curated domain knowledge described in Section IV-B. The *Online Analysis* variant (**`Direct (O)`**) receives only the tool descriptions and workflow guidance, and must infer database structure through exploratory SQL queries. The final variant (**`Direct (M)`**) keeps the same setup as **`Direct`** but replaces GPT-5 with GPT-5-mini (a weaker, lower-cost model).

We also report two *Plan-and-Execute* configurations. The plain approach (**`Plan-Execute`**) uses upfront planning, whereas the *iterative* approach (**`Plan-Execute (I)`**) plans and executes one step at a time. Because Plan-and-Execute separates planning and execution into distinct roles, we further include **`Plan-Execute (M-P)`** and **`Plan-Execute (M-E)`** variants, in which GPT-5-mini replaces only the planner or only the executor, respectively.

Finally, we report two *Reviewer-augmented* configurations. The plain approach (**`Plan-Execute-Review`**) is process-aware: the reviewer receives the planner/executor trace and intermediate outcomes. In the *process-blind* variant (**`Plan-Execute-Review (PB)`**), the reviewer receives only the task and the final candidate artifact.

a) ***Baseline: DS-STAR****:* We additionally include DS-STAR [19], a state-of-the-art open-source agent for general



(a) Data Extraction (b) Data Analysis (c) Insight Generation (d) Unsolvable

Fig. 2: Mean *average-of-3* (x) and mean *best-of-3* (y) scores across task types.

end-to-end data analysis, as a comparison baseline. DS-STAR implements a substantially more complex multi-agent workflow with specialized roles (Analyzer, Planner, Coder, Debugger, Router, Verifier, and Finalizer). To evaluate it on `CIODI-Bench`, we make minimal adaptations intended to remain faithful to the original design. First, because DS-STAR assumes a Python-only environment, we expose both our SQL query tool and our code-execution tool to the Coder and Debugger roles. Second, because DS-STAR's Analyzer is designed to explore and define the data schema before task execution, we evaluate both an *Online Analysis* variant (**`DS-Star (O)`**) with the Analyzer enabled and a variant in which the Analyzer is replaced by our curated context (**`DS-Star`**). Finally, because DS-STAR does not include explicit task rejection, we add a one-shot *SolvabilityChecker* role invoked before execution to classify tasks as solvable or not. We cap the built-in refinement parameter at 20 rounds to match the original work.

B. RQ1: Performance across task types

We begin with task type because it provides the cleanest view of the central thesis that the dominant reliability bottleneck shifts with task structure. Figure 2 reports results for each task type (*data extraction*, *data analysis*, *insight generation*, and *unsolvable tasks*). Using `CIODI-Bench`'s reliability metrics, we plot the *best-of-3* score of each agent on the y-axis and the *average-of-3* score on the x-axis. Points closer to the upper-right are therefore both more accurate and more stable.

Data Extraction. Figure 2a isolates the regime in which semantic grounding is the dominant bottleneck. A representative comparison makes the gap concrete as **`Direct`** with curated context reaches an *average-of-3* of 93.5%, while **`Direct (O)`** reaches only 73%, a gap of 20.5 points. This indicates that once the relevant schema semantics are supplied, agents reliably produce accurate results. In contrast, the substantially lower *average-of-3* results of both *Online Analysis* variants (**`Direct (O)`** and **`DS-Star (O)`**) indicate brittleness driven by upstream semantic mistakes, such as misinterpreting a column's meaning or assuming an incorrect linkage between identifiers, which propagates through otherwise plausible pipelines.

The behavior of **`DS-Star (O)`** sharpens this point. Its design allocates a dedicated *Analyzer* to summarize tables one at a time under fresh context before passing those summaries to the planner. Yet, DS-STAR does not improve meaningfully over **`Direct (O)`**, suggesting that additional agent effort spent on self-derived schema understanding does not reliably overcome the core problem and plausible-but-wrong semantic assumptions remain burdensome without vetted ground truth.

Finally, among curated-context methods, substitutions with the less capable GPT-5-mini remain competitive on extraction, further indicating that, in this regime, model capability is less important than domain-relevant semantic context.

Data Analysis. Figure 2b shows performance on *Data Analysis* tasks. As in Data Extraction, the two Online Analysis variants remain outliers, reinforcing that unresolved schema semantics upstream still degrade downstream execution. What changes in this regime is not the need for grounding, but the value of orchestration. Quantitatively, **`Plan-Execute`** raises mean *average-of-3* from 51.5% for **`Direct`** to 56.9%, and **`Plan-Execute-Review`** pushes this further to 64.7%. However, the process-blind reviewer reaches only 53.2%.

The key difference is that many Data Analysis tasks end in a comparatively constrained output space. Although the task still requires interpretation, the final answer often reduces to one of a small set of categorical conclusions (e.g., *shared_access_is_better* or *independent_access_is_better*). That structure makes planning and verification useful because the reviewer can focus on the small set of counterfactual choices that would flip the label by rerunning a targeted set of alternative approaches.

The gap between process-aware and process-blind **`Plan-Execute-Review`** approaches further supports this interpretation. The process-aware reviewer sees the planner/executor trace and intermediate artifacts, which (i) reveals which assumptions were made and which checks were omitted, and (ii) encourages *complementary* validation

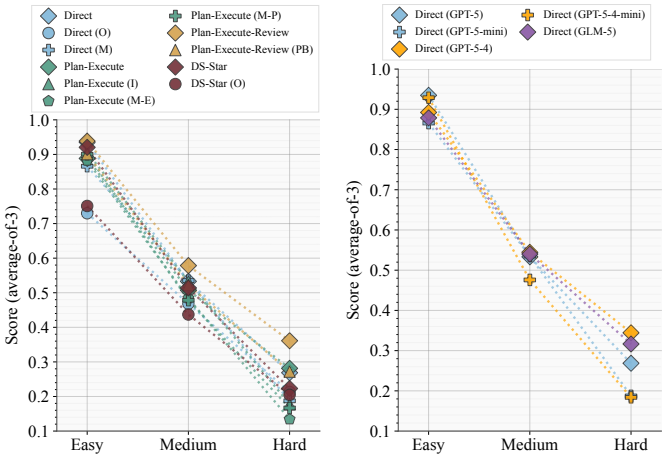
rather than replaying the same reasoning path. By contrast, a process-blind reviewer sees only the task and the final artifact, and therefore often reconstructs the same framing that produced the original result, yielding much less benefit.

Insight Generation. Figure 2c reports results on *Insight Generation* tasks. Unlike *Data Analysis*, these tasks are substantially more open-ended, requiring the agent to interpret intermediate results and synthesize free-form conclusions (e.g., detecting and characterizing trends in I/O patterns). Here the agent must decide which signals matter, synthesize evidence into an explanation, and defend a narrative under ambiguity. As a result, all configurations perform markedly worse on this task type: even the strongest setup reaches an *average-of-3* of only 45.7%. More importantly, the added orchestration that helped on Data Analysis tasks now achieves a lower score (43.7%). The plain **`Direct`** baseline performs the best, suggesting that decomposition and review are useful only when they can anchor themselves in discriminative cross-checks rather than in another layer of open-ended reasoning.

Tunnel Vision. Across task families, the mean *average-of-3* score across all approaches declines steadily from 87.4% in the most constrained setting (*Data Extraction*) to just 38.5% on the most open-ended (*Insight Generation*). Across failed episodes in both *Data Analysis* and *Insight Generation*, the most common failure mode is *tunnel vision*: agents commit early to one framing, such as a single metric, one table, or an initial hypothesis, and then complete the rest of the task within that narrowed path instead of actively testing alternatives. This is especially damaging when correctness depends on upstream modeling choices (e.g., grouping level, time window, or the operational definition of "better"), because one early mistake can still produce a complete, well-formatted artifact that looks plausible but is ultimately wrong.

Unsolvable. Unsolvable tasks (Figure 2d) provide a direct stress test of meta-reasoning about what can and cannot be concluded from the available tools and data. Across all context and orchestration settings, frontier models reliably reject these tasks, indicating that the benchmark contains enough information for a capable model to recognize when the required evidence is simply absent. However, the variants using GPT-5-mini perform substantially worse. This shows that substituting a smaller model at the *decision layer* disproportionately harms reliability. The same pattern appears on *Insight Generation* tasks, pointing to a broader regime in which open-ended reasoning over the schema and telemetry dominates. In other words, model capability matters most at upstream decision points, such as solvability judgment, plan selection, and hypothesis refinement. On *Data Analysis* tasks, this dependence is less pronounced because the constrained answer space provides some guidance about how the problem should be framed.

In total, RQ1 supports a single mechanism. Namely, as task structure shifts from constrained extraction to semi-constrained analysis and then to open-ended interpretation and abstention, the dominant reliability bottleneck moves from *semantic grounding*, to *verification tractability*, to *upstream judgment*.



(a) Trend among baseline configurations

(b) Trend among new-generation models

Fig. 3: Performance trends of *average-of-3* from Easy to Hard. Task difficulty (x) and mean *average-of-3* (y).

This bottleneck migration explains why the same intervention can be decisive for one task family yet nearly irrelevant for another. Table IV summarizes this pattern as a failure taxonomy: each task regime exposes a different dominant failure mode and therefore favors a different intervention.

C. RQ2: Performance across difficulty levels

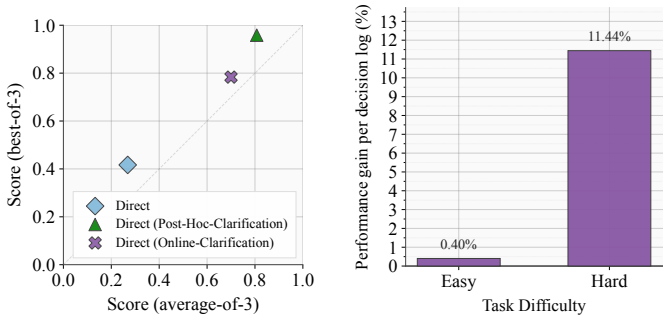
Task type explains *which* bottleneck dominates, while task difficulty shows how strongly that bottleneck constrains performance once tasks demand longer reasoning traces and more challenging domain interpretation. Figure 3a shows a monotonic decline in mean *average-of-3* across all approaches from 87.4% on easy tasks to 22.6% on hard tasks.

Hard tasks are especially informative because they often require agents to form non-trivial operational hypotheses before or during task execution. One example asks the agent to identify applications that use the MPI-IO read–modify–write (data sieving) optimization. A natural starting point is a cross-layer constraint in Darshan: because MPI-IO is implemented on top of POSIX, MPIIO activity on a file should be accompanied by concurrent POSIX activity on the same file. From this, an expert could derive a hypothesis about what read–modify–write should look like in counters: candidates should show (i) nonzero MPIIO write volume, and (ii) concurrent POSIX *reads and writes* on the same file, consistent with a read–modify–write pattern.

However, even when we provide the general fact that MPI-IO behavior is reflected in the POSIX layer, agents still fail on this task because they do not translate that fact into the concrete cross-layer conditions above. Instead, they either ignore the POSIX layer entirely and attempt detection only using MPIIO counters, or adopt other incorrect cross-layer heuristics. This explains why hard-task performance remains largely unsaturated as the bottleneck is not executing a known plan, but accurately conducting deep domain-relevant reasoning to generate the right analysis approach and conclusions.

Failure type	Most exposed scenarios	Primary mitigation
Schema semantic mismatch	Online schema discovery on data extraction and data analysis tasks	Expert-curated schema and domain context
Tunnel vision / false framing	Single-pass solving on data analysis and insight generation tasks	Process-aware review with targeted counter-checks
Hypothesis-construction failure	Weaker planners on insight generation tasks	Stronger planner model
Incorrect abstention boundary	Weaker planners on Unsolvable tasks	Explicit solvability check using stronger reasoning models

TABLE IV: Failure taxonomy distilled from repeated error analysis across `CIODI-Bench`.



(a) Impact of post-hoc and Online expert-clarification methods

(b) Average performance gain per logged/clarified decision in the Online-Clarification variant

Fig. 4: Impact of targeted expert clarification on hard tasks.

Interestingly, general advancements in model capabilities do not necessarily impact hard-task performance. As shown in Figure 3b, the performance of the newer GPT-5.4 and GPT-5.4 mini as well as state-of-the-art open-source models such as GLM-5, still remains below an admissible reliability standard compared to previous-generation GPT-5 or GPT-5-mini models.

This in turn suggests a practical deployment implication. Some failures may be resolved not by adding more agent machinery but by keeping experts in the loop to provide minimal semantic clarifications.

As a diagnostic probe, we reviewed the dominant failure mode for each hard task and added a single targeted semantic clarification that addressed that failure without revealing the answer. We report the results as the Post-Hoc-Clarification variant in Figure 4a, comparing it to the baseline (Direct-Solver) performance on hard tasks. Quantitatively, this single semantic patch raises the *average-of-3* performance from 24.7% to 80.7%.

Importantly, each clarification is limited to a single semantic point and does not reveal the final answer. For example, the expert clarification for the data sieving identification task above only adds a brief instruction about how to identify read–modify–write behavior (e.g., "use the cross-layer POSIX read+write signature on the same file as the detection rule"), which is enough to effectively convert a near-impossible task into a more routine processing task closer to the *Data Extraction* regime.

However, this kind of post-hoc failure analysis is time consuming and inherently hindsight-based, while providing such clarification in advance is difficult because we do not know which domain assumptions the agent will mishandle. Hence, an important direction for future work is to identify effective ways to keep experts in the loop for domain-specific agentic data analysis.

As an initial step in this direction, we apply a naive approach in which we instruct the agent to log any of its decisions that were not clarified in the user's prompt using a dedicated decision-logger tool. We treat each of the logged decisions as an escalation event in which an expert must review the logged decision and provide targeted clarification if needed. The results are represented as **`Direct (Online-Clarification)`** in Figure 4a and provide preliminary evidence that comparable gains may be achievable without a separate post-hoc rerun, although the current decision-logging strategy introduces substantial unnecessary escalations. Figure 4b shows this as the average performance gain per decision-logger invocation, highlighting that the value per logged decision is much higher for hard tasks than it is for easy ones. These initial results clearly suggest that more dedicated approaches should be studied, such as one in which each decision is triaged by executing possible branches and only escalating to a human expert if the branches meaningfully diverge in outcome. We consider these as important directions for future research inspired by this work.

Viewed in this context, our results delineate an automation boundary where extraction tasks with vetted semantic context are the strongest candidates for autonomous execution, while data-analysis and insight-generation tasks that require domain-specific hypothesis formation continue to benefit from expert clarification. Once a novel analysis becomes routine and its procedure has been validated, it can be encoded as a curated query, template, or dashboard instead of being repeatedly delegated to an autonomous agent.

D. RQ3: Cost–performance tradeoffs

Figure 5 reports the mean number of output tokens per task for each agent configuration on `CIODI-Bench`. We use output tokens as a comparable proxy for generation work because input-token counts and costs are strongly affected by tool-output verbosity and implementation-specific optimizations (e.g., prompt caching). At OpenAI's API pricing as of March 2026, 20k output tokens correspond to approximately \$0.20 for GPT-5 (\$10 per million output tokens) or \$0.04 for GPT-5-mini (\$2 per million). We also report the mean *average-of-3* score across all tasks for each setup.

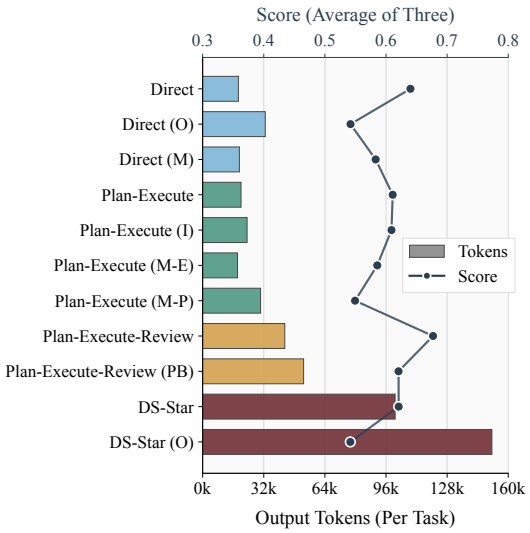


Fig. 5: Output tokens per task shown as mean generated tokens (bars) and *average-of-3* performance across all tasks (points).

The main pattern is that the interventions that improve reliability are not cost-neutral, and their cost depends strongly on where in the pipeline they operate. Across configurations, output-token usage is driven largely by how much text the agent generates in three recurring phases: (i) schema probing when context is missing, (ii) single-pass solving, and (iii) post-hoc verification or refinement (explicit review or iterative self-correction). *Online Analysis* variants generate additional tokens up front through exploratory SQL used to reconstruct schema semantics. Reviewer-augmented methods add a dedicated verification phase that either performs targeted checks (process-aware) or re-solves the task from scratch (process-blind). DS-STAR is distinct in that it intertwines solving and verification within an iterative refinement loop: a router agent may discard prior steps it deems incorrect, leading to repeated generation and higher token counts when work is redone and abandoned.

These design choices create a clear stratification in output-token usage. DS-STAR occupies the high end at over 100k mean output tokens per task, reflecting the token inefficiency of its multi-iteration refinement pipeline. Reviewer-augmented methods follow between 40k and 55k, with the process-blind reviewer typically generating more tokens because it cannot use the original trace to perform lightweight, complementary checks and instead tends to reproduce a full solution attempt. The Online Analysis Direct-Solver sits in the middle at 32k tokens because schema-discovery effort is prepended to an otherwise direct solve. In contrast, the Direct-Solver and both Plan-and-Act variants (without additional exploration or review) form the low-token regime between 15k and 30k tokens, since they devote most generation to a single end-to-end solve and therefore minimize overhead beyond core SQL and code tool use. Relative to **`Direct`**, this means process-aware review costs about 2–3$\times$ more output tokens per task, while DS-STAR costs upward of 6.5$\times$ more.

Task category	Original avg@3	Avg. @9	SD of 3-of-9 subsamples
Data Extraction	0.94	0.94	0.001
Data Analysis	0.73	0.75	0.03
Insight Generation	0.36	0.34	0.04
Unsolvable	1.00	0.96	0.05

TABLE V: Higher-K validation of the **`Direct`** configuration on 12 sampled tasks (3 tasks per category). The final column reports the standard deviation across all three-run subsets of the nine trials.

E. Scope and Validity Considerations

a) Cross-site scope and transferability.: `CIODI-Bench` is grounded in Polaris, so the results should be interpreted as controlled, single-site evidence rather than universally transferable estimates across HPC environments. Its value is precisely that this realistic setting lets us isolate the effects of semantic grounding, orchestration, and model capability; we consider broader multi-site validation as future work. However, several of our findings are tied to task structure and are therefore likely to remain consistent across facilities. First, constrained extraction tasks depend primarily on vetted schema and domain semantics. While the particular semantics differ across facilities, autonomous discovery of undocumented local conventions is likely to remain brittle. Second, tasks that have a small set of enumerable outcomes are likely to benefit from additional orchestration that enables targeted evaluation of counterfactual branches. Third, challenging open-ended analyses are likely to remain limited by upstream decisions such as problem framing, hypothesis formation, semantic interpretation, and solvability judgment, where stronger reasoning or expert clarification may be more valuable than additional orchestration. Finally, increasingly elaborate orchestration should be applied selectively, as additional agent structure generally increases generation work and provides value only when it addresses the task's dominant bottleneck.

b) Repeated-run robustness.: Our primary evaluation uses $K{=}3$ independent trials per task, which captures expected behavior but not the full tail of the outcome distribution. Accordingly, we emphasize regime-level patterns as opposed to fine-grained rankings among near-tied methods. However, to confirm whether the reported task-regime trends are sensitive to this sampling budget, we extended the **`Direct`** configuration to $K{=}9$ on a stratified subset of 12 tasks, comprising three tasks from each category. Table V compares the original *average-of-3* with the extended *average-of-9* and reports the variability across all three-run subsets of the nine trials.

Across task categories, the original *average-of-3* differs from the corresponding *average-of-9* by at most 0.04, while the standard deviation across three-run subsamples ranges from 0.001 to 0.05. These results provide additional evidence that the reported regime-level trends are not artifacts of an unusually favorable or unfavorable three-run sample.

VI. Related Work

HPC I/O telemetry analysis and observability ecosystems. A substantial body of HPC work focuses on collecting, summarizing, and analyzing I/O behavior through profilers, traces, and observability infrastructure, including Darshan, Recorder, LDMS, MELA, IOMiner, and related systems [5], [10], [30], [32], [34], [33]. PyDarshan and LASSi further illustrate how these artifacts support interactive and operational I/O analytics workflows [53], [54], while recent surveys synthesize the broader landscape of HPC I/O characterization and optimization [55], [56], [57], [58]. These works establish the realism and practical importance of telemetry analysis, but they do not evaluate LLM agents on such tasks. Closest to our setting, ION and IOAgent use LLMs to diagnose I/O issues from application traces, while STELLAR uses an LLM agent to iteratively tune a parallel file system [59], [60], [44]. In contrast, we study heterogeneous tasks over multiple operational-telemetry sources and hold the data, tools, and scoring procedure fixed to compare how context, orchestration, and model capability affect accuracy, repeated-run reliability, and cost.

Text-to-SQL and database question answering benchmarks. A large body of work evaluates natural-language-to-SQL translation over relational databases. Benchmarks such as Spider [61], [20] emphasize cross-domain generalization and complex query composition, largely in a single-query setting. BIRD moves toward more realistic databases via larger scale and richer content, and introduces external evidence to better align questions with database values [15]. More recent benchmarks such as Spider 2.0 and BEAVER target enterprise-style workflows, while Dr.Spider emphasizes diagnostic robustness under perturbation [20], [21], [62]. These benchmarks substantially improve ecological validity for database interaction, but they still center SQL generation and robustness rather than end-to-end analytic workflows with downstream computation, artifact submission, and domain-specific interpretation.

End-to-end data science benchmarks for agents. A second line of work evaluates broader data science capabilities. DS-1000 focuses on Python data science code generation with executable tests [23], while DSBench targets longer, more realistic analysis/modeling tasks [26]. Benchmarking Data Science Agents expands evaluation to the data science lifecycle via DSEval [22]. Related benchmarks evaluate tool-using agents and multi-step pipelines, including DA-Code [51], DABStep [24], KramaBench [25], and BLADE [27]. Collectively, these efforts push beyond isolated subtasks, but they typically operate in general domains and at scales that fit a file-centric (often Python-only) workflow, reducing the need for significant reasoning over domain-specific semantics.

Standalone data analyst agents and reliability-oriented evaluation. In parallel, several systems propose end-to-end data analyst agents that combine planning, tool use, and self-correction, including DS-STAR [19] and other decomposition/verification-based approaches [18], [45]. Recent work also explores fine-tuning for autonomous data analysis beyond hand-crafted agent pipelines [48] in general domains. Reliability-focused evaluation has also begun to receive more attention: Reducing Tool Hallucination via Reliability Alignment studies failure modes in tool use [63], and CORE-Bench evaluates agents on computational reproducibility tasks drawn from published scientific work [64]. These efforts reinforce the importance of trustworthy evaluation, but they do not provide a realistic, domain-specific substrate for studying reliability on HPC I/O telemetry analysis or for disentangling the effects of context, orchestration, and model capability under controlled conditions.

VII. Conclusion

In this work, we introduced `CIODI-Bench` and used it as a substrate to conduct a controlled study of tool-using LLM agents for realistic HPC I/O telemetry analysis over production Polaris data. Across 49 expert-curated tasks, the central result is that the dominant reliability bottleneck shifts with task structure: constrained extraction is limited mainly by semantic grounding, semi-constrained analysis benefits from process-aware verification when answers admit targeted falsification, and open-ended or hard tasks remain bottlenecked by upstream judgment, where stronger models help more than additional orchestration.

The practical implications are also clear. Prioritizing vetted schema and domain context drives reliability up and cost down, adding lightweight review mechanisms only where solutions are enumerable can benefit reliability, and keeping experts in the loop via dedicated mechanisms is still critical to enable highly reliable analysis results on domain-specific reasoning tasks. However, how to effectively involve experts will require substantial exploration.

Acknowledgments

We sincerely thank the reviewers for their valuable feedback. This work was supported in part by the National Science Foundation (NSF) under grants CCF-2521613 and CCF-2412345. This effort was also supported in part by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research (ASCR) under contract number AC02-06CH11357 with Argonne National Laboratory. The Authors acknowledge the National Artificial Intelligence Research Resource (NAIRR) Pilot for contributing to this research result.

References

[1] T. Hoefler, M. Copik, P. Beckman, A. Jones, I. Foster, M. Parashar, D. Reed, M. Troyer, T. Schulthess, D. Ernst, and J. Dongarra, "Xaas: Acceleration as a service to enable productive high-performance cloud computing," *Computing in Science & Engineering*, vol. 26, no. 3, pp. 40–51, 2024.

[2] V. Kumar, "High performance data mining - application for discovery of patterns in the global climate system," pp. 4–4, Springer Berlin Heidelberg.

[3] N. Wedi, P. Bauer, I. Sandu, J. Hoffmann, S. Sheridan, R. Cereceda, T. Quintino, D. Thiemert, and T. Geenen, "Destination earth: High-performance computing for weather and climate," *Computing in Science & Engineering*, vol. 24, no. 6, pp. 29–37, 2022.

[4] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo, A. Dal Corso, S. de Gironcoli, S. Fabris, G. Fratesi, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A. P. Seitsonen, A. Smogunov, P. Umari, and R. M. Wentzcovitch, "Quantum espresso: a modular and open-source software project for quantum simulations of materials," *Journal of Physics: Condensed Matter*, vol. 21, p. 395502, sep 2009.

[5] P. Carns, R. Latham, R. B. Ross, K. Iskra, S. Lang, and K. M. Riley, "24/7 characterization of petascale i/o workloads," *2009 IEEE International Conference on Cluster Computing and Workshops*, pp. 1–10, 2009.

[6] P. Carns, K. Harms, R. Latham, and R. Ross, "Performance analysis of darshan 2.2.3 on the cray xe6 platform.," 2012.

[7] S. Snyder, P. Carns, K. Harms, R. Ross, G. Lockwood, *et al.*, "Modular hpc i/o characterization with darshan," in *Proceedings of the 5th Workshop on Extreme-Scale Programming Tools (ESPT)*, IEEE, 2016.

[8] R. Wang, S. Snyder, D. Benjamin, Z. Dong, P. Gartung, and K. Herner, "Darshan for hep applications," *EPJ Web of Conferences*, vol. 295, p. 05003, 2024.

[9] S. Snyder, P. Carns, K. Harms, R. Latham, and R. Ross, "Performance evaluation of darshan 3.0.0 on the cray xc30," 2016.

[10] C. Wang, J. Sun, M. Snir, K. Mohror, and E. Gonsiorowski, "Recorder 2.0: Efficient parallel i/o tracing and analysis," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 1–8, 2020.

[11] M. J. Brim and J. K. Lothian, "Monitoring extreme-scale lustre toolkit," 2015.

[12] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Workshop on job scheduling strategies for parallel processing*, pp. 44–60, Springer, 2003.

[13] S. Snyder, P. Carns, R. B. Ross, R. Latham, and K. Harms, "Expanding community access to real-world HPC application i/o characterization data using Darshan," in *Proceedings of the Cray User Group*, 2025.

[14] B. Paranjape, S. Lundberg, S. Singh, H. Hajishirzi, L. Zettlemoyer, and M. T. Ribeiro, "ART: Automatic multi-step reasoning and tool-use for large language models," *arXiv preprint arXiv:2303.09014*, 2023.

[15] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Cao, R. Geng, *et al.*, "Can LLM already serve as a database interface? a BIg bench for large-scale database grounded text-to-SQLs," *arXiv preprint arXiv:2305.03111*, 2023.

[16] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan, "SWE-bench: Can language models resolve real-world GitHub issues?," in *International Conference on Learning Representations (ICLR)*, 2024.

[17] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou, "Text-to-SQL empowered by large language models: A benchmark evaluation," *arXiv preprint arXiv:2308.15363*, 2023.

[18] S. Hong, Y. Lin, B. Liu, B. Liu, B. Wu, C. Zhang, D. Li, J. Chen, J. Zhang, J. Wang, L. Zhang, L. Zhang, M. Yang, M. Zhuge, T. Guo, T. Zhou, W. Tao, R. Tang, X. Lu, X. Zheng, X. Liang, Y. Fei, Y. Cheng, Y. Ni, Z. Gou, Z. Xu, Y. Luo, and C. Wu, "Data interpreter: An LLM agent for data science," in *Findings of the Association for Computational Linguistics: ACL 2025*, (Vienna, Austria), pp. 19796–19821, Association for Computational Linguistics, July 2025.

[19] J. Nam, J. Yoon, J. Chen, and T. Pfister, "DS-STAR: Data science agent via iterative planning and verification," *arXiv preprint arXiv:2509.21825*, 2025.

[20] F. Lei, J. Chen, Y. Ye, R. Cao, D. Shin, H. Su, Z. Suo, H. Gao, W. Hu, P. Yin, V. Zhong, C. Xiong, R. Sun, Q. Liu, S. Wang, and T. Yu, "Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows," 2024.

[21] P. B. Chen, F. Wenz, Y. Zhang, D. Yang, J. Choi, N. Tatbul, M. Cafarella, Ç. Demiralp, and M. Stonebraker, "Beaver: An enterprise benchmark for text-to-sql," 2024.

[22] Y. Zhang, Q. Jiang, X. XingyuHan, N. Chen, Y. Yang, and K. Ren, "Benchmarking data science agents," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, (Bangkok, Thailand), pp. 5677–5700, Association for Computational Linguistics, 2024.

[23] Y. Lai, C. Li, Y. Wang, T. Zhang, R. Zhong, L. Zettlemoyer, S. W.-t. Yih, D. Fried, S. Wang, and T. Yu, "DS-1000: A natural and reliable benchmark for data science code generation," *arXiv preprint arXiv:2211.11501*, 2022.

[24] A. Egg, M. I. Goyanes, F. Kingma, A. Mora, L. von Werra, and T. Wolf, "Dabstep: Data agent benchmark for multi-step reasoning," 2025.

[25] E. Lai, G. Vitagliano, Z. Zhang, O. Chabra, S. Sudhir, A. Zeng, A. A. Zabreyko, C. Li, F. Kossmann, J. Ding, J. Chen, M. Markakis, M. Russo, W. Wang, Z. Wu, M. J. Cafarella, L. Cao, S. Madden, and T. Kraska, "Kramabench: A benchmark for ai systems on data-to-insight pipelines over data lakes," 2025.

[26] L. Jing, Z. Huang, X. Wang, W. Yao, W. Yu, K. Ma, H. Zhang, X. Du, and D. Yu, "DSBench: How far are data science agents from becoming data science experts?," *arXiv preprint arXiv:2409.07703*, 2024.

[27] K. Gu, R. Shang, R. Jiang, K. Kuang, R.-J. Lin, D. Lyu, Y. Mao, Y. Pan, T. Wu, J. Yu, *et al.*, "BLADE: Benchmarking language model agents for data-driven science," *arXiv preprint arXiv:2408.09667*, 2024.

[28] Argonne Leadership Computing Facility, "ALCF Public Data: Polaris." Data catalog web page. Accessed: 2026-01-22.

[29] Z. Zhu, L. Derstroff, and S. Neuwirth, "Benchmarking darshan and recorder for hpc i/o profiling and tracing," in *2025 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops)*, pp. 1–6, 2025.

[30] A. M. Agelastos, B. A. Allan, J. Brandt, T. Cass, A. G. Gentile, M. Grondona, I. Karlin, M. Larson, J. Livingston, T. McLarty, G. Monaco, K. Mohror, M. Ricardo, G. M. Shipman, B. Wallace, P. Widener, J. Zavala, and T. O. Tucker, "The lightweight distributed metric service: A scalable infrastructure for continuous monitoring of large scale computing systems and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14)*, pp. 154–165, IEEE Press, 2014.

[31] C. Egersdoerfer, M. H. Rashid, D. Dai, B. Fang, and N. R. Tallent, "Understanding and predicting cross-application I/O interference in HPC storage systems," in *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1330–1339, IEEE, Nov. 2024.

[32] F. Shilpika, B. Lusch, M. Emani, V. Vishwanath, M. E. Papka, and K.-L. Ma, "Mela: A visual analytics tool for studying multifidelity hpc system logs," in *2019 IEEE/ACM Industry/University Joint International Workshop on Data-center Automation, Analytics, and Control (DAAC)*, pp. 13–18, 2019.

[33] J. L. Bez, H. Tang, B. Xie, D. Williams-Young, R. Latham, R. Ross, S. Oral, and S. Byna, "I/o bottleneck detection and tuning: Connecting the dots using interactive log analysis," in *2021 IEEE/ACM Sixth International Parallel Data Systems Workshop (PDSW)*, pp. 15–22, 2021.

[34] T. Wang, S. Snyder, G. Lockwood, P. Carns, N. Wright, and S. Byna, "Iominer: Large-scale analytics framework for gaining knowledge from i/o logs," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 466–476, 2018.

[35] A. Barry, J. Brandt, A. Gentile, C. J. Morrone, E. Roman, A. Scott, K. Shoga, and T. Tucker, "Evaluating and influencing extreme-scale monitoring implementations," May 2023. LLNL-CONF-847852. Presented at Cray User Group (CUG) 2023, Helsinki, Finland, May 7–11, 2023. Dated April 19, 2023.

[36] B. Schwaller, J. M. Brandt, K. Shoga, and T. Tucker, "Evolving large scale HPC monitoring & analysis to track modern dynamic environments," (United States), Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), Sept. 2024. OSTI ID: 2588319; Report No.: SAND2024-12031C.

[37] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, S. Thomas, G. Brain, B. McGrew, I. Sutskever, and W. Zaremba, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[38] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, T. Hubert, *et al.*, "Competition-level code generation with AlphaCode," *Science*, vol. 378, no. 6624, pp. 1092–1097, 2022.

[39] A. Prasad, E. Stengel-Eskin, J. C.-Y. Chen, Z. Khan, and M. Bansal, "Learning to generate unit tests for automated debugging," in *Conference on Language Modeling (COLM)*, 2025.

[40] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, W. X. Zhao, Z. Wei, and J.-R. Wen, "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, no. 6, p. 186345, 2024.

[41] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in *The Eleventh International Conference on Learning Representations (ICLR)*, 2023.

[42] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," 2023.

[43] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez, "Gorilla: Large language model connected with massive APIs," in *Advances in Neural Information Processing Systems (NeurIPS)* (A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, eds.), vol. 37, 2024.

[44] C. Egersdoerfer, P. Carns, S. Snyder, R. Ross, and D. Dai, "Stellar: Storage tuning engine leveraging llm autonomous reasoning for high performance parallel file systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '25, (New York, NY, USA), p. 1252–1266, Association for Computing Machinery, 2025.

[45] M. Sun, R. Han, B. Jiang, H. Qi, D. Sun, Y. Yuan, and J. Huang, "Lambda: A large model based data agent," *Journal of the American Statistical Association*, pp. 1–13, 2025. Published online: 17 Jul 2025.

[46] F. Granado, R. Lotufo, and J. Pereira, "Raise: Reasoning agent for interactive sql exploration," in *Anais do Simpósio Brasileiro de Tecnologia da Informação e da Linguagem Humana (STIL 2025)*, (Porto Alegre, RS, Brasil), pp. 170–181, SBC, 2025.

[47] Z. Wang, R. Zhang, Z. Nie, and J. Kim, "Tool-assisted agent on sql inspection and refinement in real-world scenarios," *arXiv preprint arXiv:2408.16991*, 2024. Often referred to as Tool-SQL in later work.

[48] S. Zhang, J. Fan, M. Fan, G. Li, and X. Du, "Deepanalyze: Agentic large language models for autonomous data science," *arXiv preprint arXiv:2510.16872*, 2025.

[49] Z. You, Y. Zhang, D. Xu, Y. Lou, Y. Yan, W. Wang, H. Zhang, and Y. Huang, "Datawiseagent: A notebook-centric LLM agent framework for adaptive and robust data science automation," in *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing* (C. Christodoulopoulos, T. Chakraborty, C. Rose, and V. Peng, eds.), (Suzhou, China), pp. 1099–1123, Association for Computational Linguistics, Nov. 2025.

[50] S. Snyder, A. L. C. Facility, P. Carns, and W.-k. Liao, "Alcf polaris darshan log collection," Oct. 2025.

[51] Y. Huang, J. Luo, Y. Yu, Y. Zhang, F. Lei, Y. Wei, S. He, L. Huang, X. Liu, J. Zhao, and K. Liu, "DA-code: Agent data science code generation benchmark for large language models," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing* (Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, eds.), (Miami, Florida, USA), pp. 13487–13521, Association for Computational Linguistics, Nov. 2024.

[52] L. E. Erdogan, H. Furuta, S. Kim, N. Lee, S. Moon, G. Anumanchipalli, K. Keutzer, and A. Gholami, "Plan-and-act: Improving planning of agents for long-horizon tasks," in *Forty-second International Conference on Machine Learning*, 2025.

[53] J. Luettgau, S. Snyder, T. Reddy, N. Awtrey, K. Harms, J. L. Bez, R. Wang, R. Latham, and P. Carns, "Enabling agile analysis of I/O performance data with PyDarshan," in *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage and Analysis*, 2023.

[54] K. Sivalingam, H. Richardson, A. Tate, and M. Lafferty, "Lassi: Metric based i/o analytics for hpc," 2019.

[55] J. L. Bez, S. Byna, and S. Ibrahim, "I/o access patterns in hpc applications: A 360-degree survey," *ACM Comput. Surv.*, vol. 56, Sept. 2023.

[56] H. Ather, J. L. Bez, C. Wang, H. Childs, A. D. Malony, and S. Byna, "Parallel i/o characterization and optimization on large-scale hpc systems: A 360-degree survey," 2024.

[57] M. H. Rashid, X. Li, Y. He, F. S. Bao, and D. Dai, "DIAL: Decentralized I/O autotuning via learned client-side local metrics for parallel file system," in *2025 IEEE 25th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pp. 1–4, IEEE, May 2025.

[58] M. H. Rashid and D. Dai, "AdapTBF: Decentralized bandwidth control via adaptive token borrowing for HPC storage," in *2025 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 775–788, IEEE, June 2025.

[59] C. Egersdoerfer, A. Sareen, J. L. Bez, S. Byna, and D. Dai, "ION: navigating the HPC I/O optimization journey using large language models," in *Proceedings of the 16th ACM Workshop on Hot Topics in Storage and File Systems, HotStorage 2024, Santa Clara, CA, USA, July 8-9, 2024*, ACM, 2024.

[60] C. Egersdoerfer, A. Sareen, J. L. Bez, S. Byna, D. Xu, and D. Dai, "Ioagent: Democratizing trustworthy HPC I/O performance diagnosis capability via llms," in *IEEE International Parallel and Distributed Processing Symposium, IPDPS 2025, Milano, Italy, June 3-7, 2025*, pp. 322–334, IEEE, 2025.

[61] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, eds.), (Brussels, Belgium), pp. 3911–3921, Association for Computational Linguistics, Oct.-Nov. 2018.

[62] S. Chang, J. Wang, M. Dong, L. Pan, H. Zhu, A. H. Li, W. Lan, S. Zhang, J. Jiang, J. Lilien, S. Ash, W. Y. Wang, Z. Wang, V. Castelli, P. Ng, and B. Xiang, "Dr.spider: A diagnostic evaluation benchmark towards text-to-sql robustness," 2023.

[63] H. Xu, Z. Zhu, L. Pan, Z. Wang, S. Zhu, D. Ma, R. Cao, L. Chen, and K. Yu, "Reducing tool hallucination via reliability alignment," 2024.

[64] Z. S. Siegel, S. Kapoor, N. Nagdir, B. Stroebl, and A. Narayanan, "Core-bench: Fostering the credibility of published research through a computational reproducibility agent benchmark," 2024.