DACOMP: BENCHMARKING DATA AGENTS ACROSS THE FULL DATA INTELLIGENCE LIFECYCLE

Anonymous authors
Paper under double-blind review

ABSTRACT

Real-world enterprise data intelligence workflows encompass data engineering that turns raw sources into analytical-ready tables and data analysis that convert those tables into decision-oriented insights. We introduce DAComp, a benchmark of 236 tasks that mirrors these complex workflows. Data engineering (DE) tasks require repository-level engineering on industrial schemas, including designing and building multi-stage SQL pipelines from scratch and evolving existing systems under evolving requirements. Data analysis (DA) tasks pose open-ended business problems that demand strategic planning, exploratory analysis through iterative coding, interpretation of intermediate results, and the synthesis of actionable recommendations. Engineering tasks are scored through execution-based, multi-metric evaluation. Open-ended tasks are assessed by a reliable, experimentally validated LLM-judge, which is guided by hierarchical, meticulously crafted rubrics. Our experiments reveal that even state-of-the-art agents falter on DAComp. Performance on DE tasks is particularly low, with success rates under 20%, exposing a critical bottleneck in holistic pipeline orchestration, not merely code generation. Scores on DA tasks also average below 40%, highlighting profound deficiencies in open-ended reasoning and demonstrating that engineering and analysis are distinct capabilities. By clearly diagnosing these limitations, DAComp provides a rigorous and realistic testbed to drive the development of truly capable autonomous data agents for enterprise settings. Our data and code are available at `https://anonymous.4open.science/r/DAComp-397A`.

1 INTRODUCTION

Data intelligence, the process of transforming raw and fragmented data into actionable insights, has become a cornerstone of modern enterprises. The remarkable reasoning and code generation capabilities of Large Language Models (LLMs) (OpenAI, 2025; Anthropic, 2025; Gemini, 2025) have opened new avenues for automating data intelligence tasks. LLM-based agents have demonstrated considerable promise across a wide range of applications, including text-to-SQL (Yu et al., 2018; Li et al., 2024b; Lei et al., 2024), software engineering (Jimenez et al., 2023; Chan et al., 2024), and general computer control (Zhou et al., 2024; Xie et al., 2024; Wei et al., 2025). However, the advancement of these agents into enterprise data intelligence remains constrained by the absence of benchmarks that faithfully reflect real-world complexity.

This gap between existing benchmarks and real enterprise practice calls for a benchmark that evaluates agents along two distinct axes: *Hard* (engineering realism) and *Soft* (analytical openness). The *Hard* axis reflects the capacity for systematic large-scale code implementation, similar to the responsibilities of data engineers. For example, this means not only generating a single SQL query but also orchestrating and evolving complex data workflows under changing requirements. The *Soft* axis reflects the capacity for strategic reasoning, aligning more closely with the role of data analysts. For example, this involves facing an open-ended business question, planning multi-step analytical workflows, synthesizing insights across analytical results, and crafting decision-oriented reports. Most benchmarks fail to capture these two key dimensions. They reduce complex engineering to isolated code snippet generation, missing the *Hard* axis, and reduce open-ended analysis to deterministic answers, missing the *Soft* axis.

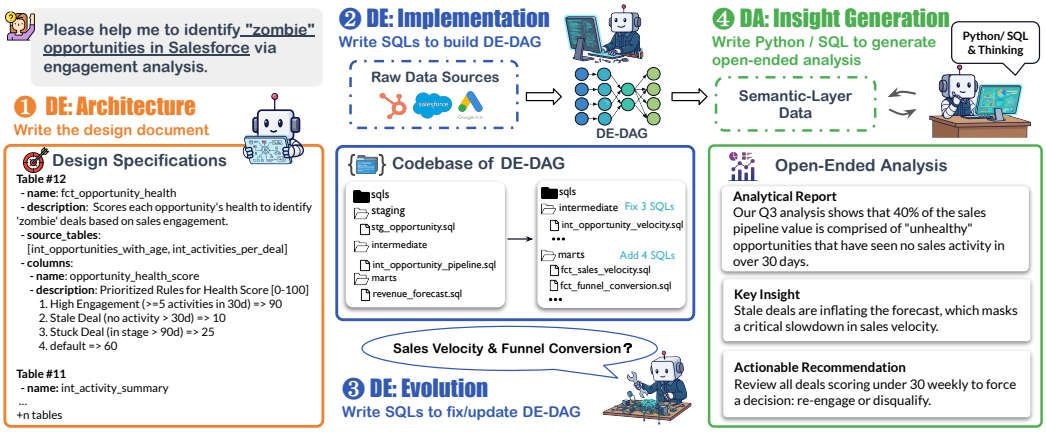

Figure 1: DAComp aims to evaluate LLMs on full-lifecycle data intelligence workflows, encompassing repository-level data engineering (*DE*) and open-ended data analysis (*DA*).

To fill this gap, we present **DAComp**, benchmarking agents on full lifecycle data intelligence tasks, as illustrated in Fig. 1. **DAComp-DE** is the first to introduce repository-level data engineering tasks where agents must orchestrate multi-layered data workflows by generating a DAG on complex enterprise schemas. It includes three distinct task types: (1) ***DE-Architecture*** tasks focus on the high-level planning of detailed engineering specifications. (2) ***DE-Implementation*** tasks require agents to build multi-stage data pipelines from scratch; (3) ***DE-Evolution*** tasks challenge them to modify existing systems in response to new requirements; and Both DE-Impl and DE-Evol tasks are demanding, often requiring large-scale code changes that involve over $4,000$ lines of code across more than 30 files, mirroring real-world engineering workloads. **DAComp-DA** is the first to pioneer real-world, open-ended data analysis. In these scenarios, agents are presented with complex questions over downstream analytical data. Unlike prior work with deterministic answers (Jing et al., 2024; Lei et al., 2024), the tasks resemble real analyst settings: agents must write SQL/Python to aggregate, compute, and analyze intermediate results in order to generate insights and reports, thereby emphasizing both the rigor of analytical precision and the practical utility for human decision-making.

The evaluation methods of such complex tasks are non-trivial. For deterministic DE-Impl and DE-Evol tasks, we adopt an execution-based method to systematically evaluate the repo-level code generation performance. The open-ended DA and DE-Arch tasks are assessed by an LLM judge (Li et al., 2024a), whose evaluation is guided by our novel rubric framework. Instead of relying on a single answer key, this framework explicitly defines and assesses multiple valid solution paths for each open-ended problem, enabling a robust, multifaceted assessment that rewards diverse analytical strategies. The reliability of this LLM judge has been confirmed through rigorous validation experiments, which show strong agreement with human experts.

Our experiments on DAComp underscore a significant challenge for current models: even state-of-the-art agents falter when confronted with its enterprise-level complexity. In DE tasks, agent capabilities are pushed to their limits, with average scores below 40% and strict success rates under 10%, revealing a critical gap in real repository-level engineering capabilities. In the same vein, agents also exhibit poor performance on open-ended problems requiring autonomous planning. Performance on DA tasks plummets to below 50% for most models, with only a few proprietary systems demonstrating more robust analytical skills. Ultimately, progress in data agents demands a shift from mere code accuracy to the nuanced capabilities—planning, open-ended reasoning, and systematic synthesis—required to deliver insights that are both analytically rigorous and strategically actionable. By providing this rigorous, realistic testbed, DAComp aims to shift the focus of data agent development from isolated skills to the integrated, full-lifecycle capabilities required in the real-world scenarios.

2 Benchmark Construction

In this section, we introduce the definition, annotation pipeline, evaluation methods and statistics.

2.1 TASK DEFINITION

To bridge this gap, we design tasks that evaluate data agents on real-world challenges. Specifically, we assess their ability to act as data engineers performing *repository-level data engineering* and as data analysts navigating *open-ended data analysis*, as depicted in Fig. 1.

DAComp-DE. An agent π^{de} is tasked with handling the full DE lifecycle including architecture, implementation, and evolution. Formally, the process is modeled as $(\mathcal{S}, \mathcal{C}_\star) = \pi^{de}(\mathcal{Q}_{\text{de}}, \mathcal{C}_0, \mathcal{B})$, where $\mathcal{Q}_{\text{de}}$ is the initial high-level requirement, $\mathcal{S}$ denotes the engineering specification (e.g., a Data Contract), $\mathcal{B}$ is the database and $\mathcal{C}_\star$ is the final DE repository. This unified capability is evaluated across three task types: *(1) DE-Arch* : Given a high-level requirement $\mathcal{Q}_{\text{de}}$ and an initial repository $\mathcal{C}_0$, this task evaluates the agent's ability to produce the engineering specification $\mathcal{S}$. *(2) DE-Impl* : Given a detailed specification $\mathcal{S}$ and an empty repository ($\mathcal{C}_0 = \varnothing$), this task evaluates the agent's ability to implement the DE repository $\mathcal{C}_\star$ from scratch. *(3) DE-Evol* : Given an existing repository $\mathcal{C}_0$ and a new specification $\mathcal{S}$, this task evaluates the agent's ability to update the repository into $\mathcal{C}_\star$.

DAComp-DA. Given an analysis-ready data $\mathcal{D}$ (semantic layer) and an open-ended question $\mathcal{Q}_{\text{da}}$, an agent with policy π^{da} produces analysis artifacts $\mathcal{O} = \pi^{da}(\mathcal{Q}_{\text{da}}, \mathcal{D})$ (e.g., analytical reports, key insights and actionable recommendations). This task is inherently open-ended, as a single question may be approached through multiple valid analytical paths, without a fixed standard answer.

2.2 EVALUATION METRICS

LLM-judge with hierarchical rubrics and GSB scoring. The LLM judge evaluates outputs $\mathcal{O}$ along five dimensions: *Completeness*, *Accuracy*, *Insightfulness*, *Readability*, and *Analytical depth* (see App. A.3.1). The hierarchical rubric assesses the first three, while the Good–Same–Bad (GSB) score (Zheng et al., 2023) covers the latter two. As shown in Fig. 2, the rubric ($\mathcal{R}$) decomposes a question $\mathcal{Q}$ into requirements and subreqirements. Each subrequirement admits multiple valid solution paths, each path carrying its own rubric items (colored leaf nodes). Human experts enumerate these paths and merge equivalent solutions in a single path. For scoring, the LLM judge selects the best-matching path for each sub-requirement, applies only that path's items, then aggregates scores bottom-up. This design accommodates diverse correct approaches without penalizing method choice. We show a detailed rubric example for the penetration and profitability analysis in Tab. 6, with a discussion of the path enumeration scheme provided in App.G.1. The rubric score is a normalized, weighted sum of satisfied items: $\text{Score}_{\text{rubric}}(\mathcal{O}, \mathcal{R}) = \frac{\sum_{k=1}^{N} s_k}{\sum_{k=1}^{N} w_k}$, $s_k = \Lambda(c_k, \mathcal{O}) \in [0, w_k]$. For the Good-Same-Bad (GSB), the LLM judge only compares the final analytic results against five pre-provided baseline reports, guided by the dedicated rubrics for these axes, yielding the score: $\text{Score}_{\text{gsb}}(\mathcal{O}, \mathcal{O}_{base}) = \frac{\max(0, |G|-|B|)}{|G|+|S|+|B|}$. The final score for a DA task is a weighted combination of these two components: $\text{Score}_{\text{da}} = \alpha \cdot \text{Score}_{\text{rubric}} + (1-\alpha) \cdot \text{Score}_{\text{gsb}}$. The open-ended DE-Arch tasks are assessed similarly, though they employ a standard, non-hierarchical rubric and do not incorporate the GSB component. Further details are provided in App. A.

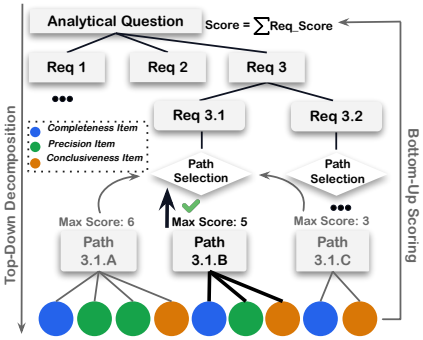

Figure 2: Details of hierarchical rubrics.

Execution-based evaluation for deterministic tasks. DE-Impl and DE-Evol tasks are evaluated with three execution-based metrics of increasing strictness: (1) the partial credit *Component Score (CS)*, $\text{CS}_{\text{DE-Impl/Evol}} = \sum_j w_j s_j$, which evaluates each node *in isolation* (using gold-standard upstream inputs) to measure total component-level SQL generation; (2) the *Cascading Failure Score (CFS)*, which evaluates nodes *sequentially along the DAG* and nullifies a node's score if any upstream dependency is incorrect, thus measuring end-to-end data integrity; and (3) the strict *Success Rate (SR)*, $\text{SR}_{\text{DE-Impl/Evol}} = \mathbb{I}[\forall j : s_j = 1]$, which requires every single component to be perfect. This suite of metrics is crucial for diagnosing the primary bottleneck: the gap between an agent's component-level generation and its ability to perform holistic pipeline orchestration. Further details are provided in App. A.1.

2.3 ANNOTATION PIPELINE

DAComp is constructed by 8 experts through a rigorous pipeline to ensure realism, quality, and consistency. Further details and examples are provided in App. E.

1) Data collection. The benchmark is grounded in permissively licensed assets (e.g., Apache-2.0, MIT). For the DE task, we collect 73 enterprise-scale SaaS schemas with data transformation projects, averaging 400 columns each, and populate them with large-scale, relationally consistent synthetic data (see App. E). For the DA task, we curate 100 complex databases from the Web and supplement them with analytical modeling layers derived from DE-transformed data.

2) Task design. At this stage, we generate the DAComp questions. For DA , annotators first draft 8 open-ended analytical questions per analysis-ready table. Five annotators then vote based on realism and difficulty, and the top 2 are retained. For DE-Evol , practicing data engineers author new business requirements aligned with enterprise scenarios and professional standards. For DE-Impl , we reverse engineer selected SaaS transformation projects into a single `data_contract.yaml`, capturing the full DAG and semantics. For DE-Arch , starting from the analytics layer of DE-Impl and DE-Evol examples, DA annotators propose 5 candidate business requirements per project, from which a data engineer selects 1 feasible yet challenging requirement.

3) Evaluation construction. We design evaluation protocols for each task. For *DA*, annotators build hierarchical rubrics as described in §2.2, with at least 3 annotators annotate each question, followed by alignment discussion to resolve discrepancies. For the *GSB* protocol, experienced data analysts author shared scoring criteria, and baseline reports are created by combining outputs from multiple LLMs. A critical aspect of this rubric design is the enumeration of valid solution *Paths*, a process governed by three key principles: (i) ensuring Paths represent distinct, methodologically-sound strategies, not incremental steps; (ii) validating deterministic outputs against programmatically calculated and verifiable anchor values; and (iii) utilizing methodology-based soft constraints to fairly evaluate valid but unenumerated solution paths. (see examples in App.C.4, discussion in App.G.1). To ensure the comprehensiveness of our rubric, we perform a validation step: we sample outputs from five diverse LLMs and confirm that our enumerated `paths` can account for all observed solution strategies, which minimizes the risk of false negatives by ensuring that valid but unanticipated solutions are not unfairly penalized. For DE-Impl and DE-Evol , solutions are deterministic: we implement execution scripts to automatically validate outputs against gold repositories, assigning partial credit at the node/layer level to capture step-wise correctness.

Table 1: Comparison of DAComp with other agent benchmarks, highlighting key differences in task scope, task paradigm, and evalution method.

Benchmark	Field	# Tasks	Repo-Level	# Cols/ Schema	Code Scale (LOC)	Primary Output	Open-ended	Evaluation Method
Agentic Benchmarks								
SWE-Bench (Jimenez et al., 2023)	Software Engineering	2,294	✓	N/A	32.8	Code Patch	✗	Execution-based
WebArena (Zhou et al., 2024)	Web Navigation	812	✓	N/A	N/A	Actions	✗	Execution-based
OSWorld (Xie et al., 2024)	Computer Control	369	✓	N/A	N/A	Actions	✗	Execution-based
BrowserComp (Wei et al., 2025)	Deep Research	2,000	✓	N/A	N/A	Answer	✓	Objective
Data Agent Benchmarks								
DS-1000 (Lai et al., 2023)	Data Science	1,000	✗	N/A	3.6	1 Script	✗	Execution-based
BIRD (Li et al., 2024b)	Text-to-SQL	12,751	✗	54	23.5	1 SQL	✗	Execution-based
Spider 2.0 (Lei et al., 2024)	Text-to-SQL	632	✗	320	104.6	1 SQL	✗	Execution-based
BIRD-CRITIC (Li et al., 2025)	SQL Debugging	1,100	✗	54	50∼70	1 SQL	✗	Execution-based
DA-Code (Huang et al., 2024)	Data Science	500	✗	50 ∼ 100	85	1 Script	✗	Objective
DSBench (Jing et al., 2024)	Data Science	540	✗	27	10∼20	N Scripts	✗	Objective
KramaBench (Lai et al., 2025)	Data Science Pipelines	104	✗	13	50∼100	N Scripts	✓	LLM-judge
BLADE (Gu et al., 2024)	Data Analysis	259	✗	10 ∼ 12	70∼80	Report	✓	LLM-judge
DABStep (Egg et al., 2025)	Data Analysis	450	✗	10 ∼ 12	100	Answer	✗	Objective
DAComp (Ours)	**Data Engineering & Data Analysis**	**236**	✓	**412**	**∼2,000**	**Doc + Report N SQL/Script**	**Both**	**Execution-based & LLM-judge(rubrics)**

2.4 DATASET STATISTICS

We present a statistical analysis of DAComp, highlighting its main features in comparison with prior datasets in Tab. 1, and providing more detailed characteristics in Tab. 2.

DAComp-DE quantifies enterprise-scale engineering complexity. The statistics for DAComp-DE underscore its large scale and complexity—defined by its repo-level paradigm, schemas averag-

ing 412 columns, and solutions requiring over 2,000 lines of code—setting it apart from prior data agent benchmarks. Unlike benchmarks that focus on generating isolated scripts, DAComp introduces tasks on industrial schemas with an average of 32 tables and 412 columns. The engineering effort required is substantial. Implementation tasks involve building entire pipelines from scratch, averaging 4,612 lines of code across 43 distinct files. Similarly, Evolution tasks simulate realistic maintenance with edits averaging 1,718 LOC across 13 files, agents need to manage data transformation across a multi-layered data model (e.g., staging, core, and mart).

Table 2: Key statistics for DAComp. All metrics are per-example averages, except #Total tasks.

Metric	Value	Metric	Value
Overall (DE-Arch/DE-Impl/DE-Evol/DA)		**DAComp-DE**	
#Total tasks	24 / 36 / 76 / 100	DE-Impl raw data (#Tab. / #Col.)	31.9 / 450.7
#Question Tokens	262 / 5,792 / 606 / 72	#LOC code scale (Impl / Evol)	4,612 / 1,718
DAComp-DA		#Change files (Impl / Evol)	42.7 / 13.1
Columns / Tables	71.8 / 5.1	#Change columns (Impl / Evol)	751.3 / 257.0
LOC	347	#DE-Arch rubric	18.5
Rubrics (Reqs / Sub-reqs / Paths / Items)	3.1 / 5.7 / 12.7 / 22.4	DE-Impl layer (#Staging / #Core / #Mart)	18.56 / 9.4 / 11.8
Completeness / Accuracy / Insightfulness	14% / 66% / 20%	DE-Evol table change types (#create / #edit)	4.34 / 7.97

DAComp-DA measures analytical depth and methodological diversity. The design of DAComp-DA moves beyond simple question-answering to assess deep analytical reasoning. Uniquely, DAComp evaluates both deterministic engineering and open-ended analysis, a distinction from prior benchmarks that typically focus on only one paradigm. Its open-ended nature is quantified by our hierarchical rubrics, which decompose each of the 100 DA tasks into an average of 3.1 requirements and 5.7 sub-requirements, accommodating roughly 13 valid solution paths. This methodological diversity is evaluated with a multi-faceted rubric where scoring items are weighted toward Accuracy (66%) but also reward Completeness (14%) and Insightfulness (20%). While the analytical schemas are more focused than in DE tasks (averaging 5 tables and 72 columns), the required reasoning is still complex, reflected in an average solution length of 347 lines of code—significantly longer than typical text-to-SQL or single-script data science tasks.

3 Experiments

3.1 Experimental Setup

We evaluate a suite of state-of-the-art LLMs, including open-source models like Qwen3 (Yang et al., 2025), DeepSeek-V3.1 (Liu et al., 2024), and Kimi-K2 (Team et al., 2025), as well as proprietary ones such as the Gemini (Team et al., 2023), Claude (Anthropic, 2024), and GPT (OpenAI, 2023) families. To provide context, we compare them against two baselines: a data agent baseline using Bash and file system operations, with the ability to execute Python and SQL, and the widely used OpenHands (CodeAct-Agent) framework (Wang et al., 2024). The performance of each agent is measured using the metrics detailed in §2.2. We also report two aggregate scores: the DE Score, which is the mean score across all DE tasks (using CFS for Implementation/Evolution), and the Overall Score, representing the mean across the entire benchmark. For the DA score, we use $\alpha = 0.8$ to aggregate the rubric and GSB scores, with Gemini-2.5-Flash serving as the LLM judge. Further details on the experimental setup and additional results are provided in App. B.

3.2 Main Results

Strong LLMs deliver robust performance across frameworks. As shown in Tab. 3, large proprietary models such as GPT-5 and o3 consistently achieve the highest scores across both DAComp-DE and DAComp-DA, clearly outperforming other systems. Claude-4-Sonnet also delivers competitive results, while models like Gemini-2.5-Pro and DeepSeek-V3.1 occupy the middle tier. In contrast, smaller open-source agents such as Qwen3-8B and Qwen3-235B-A22B lag significantly behind, often failing to surpass 20% overall. These patterns underscore that raw model capacity remains the dominant factor in determining benchmark performance. Nevertheless, framework design still plays a modulatory role: strong LLMs perform well regardless of orchestration, but their relative margins can shift depending on interaction stability and error-handling, and weaker models are especially sensitive to these system-level choices. This suggests that future improvements will require advances both in underlying model capability and in the frameworks that govern agent behavior.

Table 3: Performance score (%) for various agent systems on DAComp. DE performance is measured by a suite of metrics with escalating difficulty: Success Rate(SR), Cascading Failure Score(CFS), and Component Score(CS) (§2.2). More models in Tab.9.

Method	DAComp-DE								DAComp-DA	Overall Score
	Architecture	Implementation			Evolution			Total		
		SR	CFS	CS	SR	CFS	CS			
OpenHands Framework										
GPT-5	**66.44**	22.22	**51.38**	79.08	**14.47**	28.87	**53.90**	41.46	**52.06**	**45.95**
o3	64.55	**38.89**	50.17	**80.43**	13.16	**30.78**	41.88	**41.87**	48.78	44.80
Claude-4-Sonnet	63.7	19.44	48.02	68.74	11.84	28.34	41.9	39.79	51.33	44.68
Gemini-2.5-Pro	55.82	11.11	41.85	63.66	9.21	26.67	43.70	35.83	40.74	37.91
DeepSeek-V3.1	58.05	2.78	23.35	38.74	5.26	24.04	33.03	29.86	45.79	36.61
Qwen3-Coder	55.31	5.56	31.76	50.66	6.25	28.65	43.17	34.18	41.88	37.44
Qwen3-235B-A22B	49.66	2.78	6.57	6.61	5.26	18.61	21.72	20.90	31.29	25.30
Qwen3-8B	47.16	2.78	4.29	4.29	2.63	13.49	23.15	17.00	17.15	17.06
Agentic Baseline										
GPT-5	67.47	27.78	**51.50**	**77.96**	**15.7**	**27.94**	**46.08**	**41.15**	**56.03**	**47.46**
o3	**67.98**	**38.89**	40.58	61.83	10.5	24.05	43.1	36.18	47.53	40.99
Claude-4-Sonnet	60.96	22.22	47.24	75.77	13.2	25.36	45.76	37.43	51.98	43.60
Gemini-2.5-Pro	58.22	11.11	42.43	73.87	7.8	22.07	23.41	33.84	40.47	36.65
DeepSeek-V3.1	60.45	5.55	33.03	50.35	6.5	20.78	38.1	31.02	45.81	38.14
Qwen3-Coder	53.25	8.33	30.09	47.2	9.2	18.05	26.7	27.45	35.28	30.77
Qwen3-235B-A22B	47.38	8.33	14.48	23.38	3.9	16.44	23.62	21.38	40.1	29.31
Qwen3-8B	44.6	2.78	3.49	4.02	3.9	6.7	9.6	12.54	18.7	15.15

The results in Tab. 3 underscore the profound challenge of DAComp, with even the top overall scores barely reaching 50%. More crucially, the data reveals that the skills for engineering and analysis are distinct, as model rankings are inconsistent across the two domains. This divergence is clearly visible among open-source models. For instance, in the OpenHands framework, the code-specialized Qwen3-Coder achieves a respectable DE score of 34.18, outperforming DeepSeek-V3.1 (29.86). However, in the open-ended DA task, the roles are reversed: DeepSeek-V3.1's score of 45.79 is significantly higher than that of Qwen3-Coder (41.88). This inversion in rankings—where a model proficient in engineering is not necessarily the leader in analysis—provides strong empirical evidence that repository-level coding and open-ended reasoning are distinct capabilities.

3.3 Analysis of Repository-level Data Engineering

Holistic orchestration is the core bottleneck in data engineering. Across DE tasks, models plan well but struggle to execute end-to-end. Architecture scores are relatively high (e.g., GPT-5: $66 \sim 67\%$), yet strict SR for Implementation and Evolution are much lower (typically $< 30\%$). The drop from CS to CFS and then to SR is pronounced for strong models, revealing a pipeline-level orchestration bottleneck beyond single-file correctness; for example, GPT-5 (OpenHands) in Implementation falls from CS 79.08 to CFS 51.38 and SR 22.22, and in Evolution from CS 53.90 to CFS 28.87 and SR 14.47. By contrast, weaker open-source models (e.g., Qwen3-8B) exhibit very low CS (Implementation 4.29), indicating deficits already at the component level; orchestration then compounds failure but is not the sole cause. The uniformly low SR across models confirms that coordinating dependencies in a live repository—rather than generating isolated correct code—is the dominant challenge in DAComp-DE.

Medium-scale code edits are the most difficult to perform. To gain a more granular understanding, we delve into a node-level analysis, studying the scores for individual SQL file modifications (Fig. 3). We classify these modifications into two types—**editing** an existing file or **creating** a new file, and group them by the required number of lines. For **create** tasks, models like GPT-5 and Claude-4 have a clear "sweet spot" on medium-scale cre-

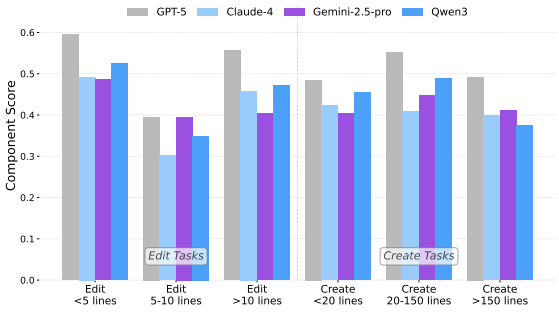

Figure 3: Component-level performance analysis.

ations ($20 - 150$ lines), while all models struggle with very large files (> 150 lines). In In contrast, **edit** tasks exhibit a non-linear difficulty trend. Contrary to intuition, medium-scale edits prove to be the most challenging. This is because minor edits are often trivial, while very large edits frequently involve repetitive, boilerplate transformations with clear logic. In contrast, medium-scale edits tend to contain the most complex and nuanced changes to business logic, aggregations, and calculations, thus posing the greatest reasoning challenge.

Analytical complexity and failure rates escalate in higher pipeline layers. Fig. 4 reveals that the difficulty of data engineering tasks escalates significantly as agents move from the initial data ingestion layer to the more complex analytical layers. The *staging* layer, focused on basic cleaning, consistently has the fewest local errors and the highest task survival rate. The challenge intensifies dramatically in the *intermediate (core)* layer. This is where the most complex business logic and entity integration occurs, and as Panel (a) shows, it is where the largest share of local errors originates. The severe impact of this difficulty is evident in Panel (b), which shows the sharpest drop in pipeline survival occurring after this stage. Finally, the *marts* layer remains highly challenging. Failures in this final stage are often a direct consequence of inheriting upstream errors from the *core* layer, with fewer than 20% of the initial tasks surviving to completion. Together, these results demonstrate a clear hierarchy of difficulty, with the analytical complexity of the *core* and *marts* layers posing a substantially greater challenge than the initial *staging* layer.

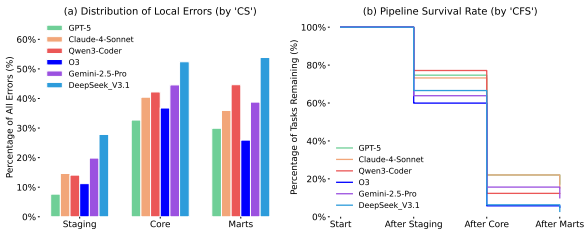

Figure 4: Contrasting local error distribution (left) with pipeline survival rate (right).

Top-performing agents exhibit stable and task-aligned interaction patterns. Fig. 5 shows the distribution of interaction turns in DE tasks. High-performing models such as GPT-5 maintain moderate turn counts with compact variance across both Implementation and Evolution settings, reflecting efficient yet sufficiently thorough reasoning. In contrast, weaker models like Qwen3 either generate excessively long and volatile traces in Implementation or display unusually short traces in Evolution, where premature termination often corresponds to incorrect or incomplete outputs. These patterns indicate that stable and centered turn distributions are more characteristic of effective agents than simply minimizing the number of turns.

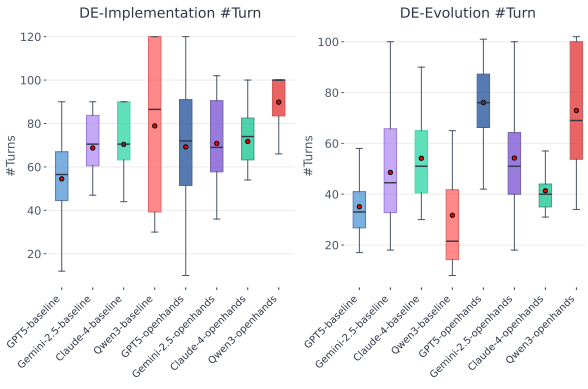

Figure 5: Turn counts on DE tasks.

Error analysis. Failures in DAComp occur at two levels. Single-File-Level Failures (Fig. 12) are typically not syntactic but rather semantic, where executable code implements flawed business logic (e.g., incorrect join keys or metric definitions). More critical are Pipeline-Level Failures(Figs. 13 and 14), which demand holistic, cross-file reasoning. These include *Dependency Graph Breakage*, *Interface Contract Violations* where changes fail to propagate downstream, and cascading *Data Integrity Errors*. Success thus requires orchestrating a correct and consistent data repository, not just generating isolated code snippets. Details are shown in App.D.

3.4 Analysis of Open-ended Data Analysis Tasks

Performance across dimensions. Our analysis of model capabilities across five analytical dimensions, defined by our rubrics and visualized in Fig. 6, reveals distinct performance profiles. The results show GPT-5 consistently outperforms other models across all axes. While Claude and DeepSeek perform strongly on foundational dimensions like Completeness and Accuracy, Qwen3 unexpectedly achieves the highest Readability. Notably, GPT-5 leads all models by a large margin in Analytical Depth, and proprietary models generally demonstrate stronger Insightfulness than open-source counterparts. This dimension-wise characterization allows for a precise analysis of each model's unique strengths and weaknesses.

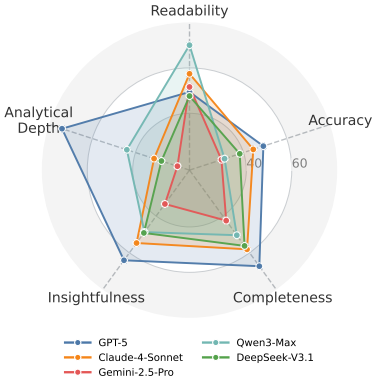

Figure 6: DA performance comparison across five dimensions.

Performance across analytical objectives. To investigate how performance correlates with the nature of the analytical task, we manually classify each DA task into five categories based on its primary objective: Descriptive, Diagnostic, Strategic, Pattern Recognition, and Profiling (see App. C.4 for definitions). As shown in Fig. 7, this classification reveals a distinct performance hierarchy. Agents excel at concrete Descriptive tasks (*what happened?*), but their scores drop sharply on more abstract Diagnostic (*why did it happen?*) and Strategic (*what should we do?*) tasks. This confirms that these more complex objectives are not only more challenging but also serve as better differentiators of advanced model capabilities.

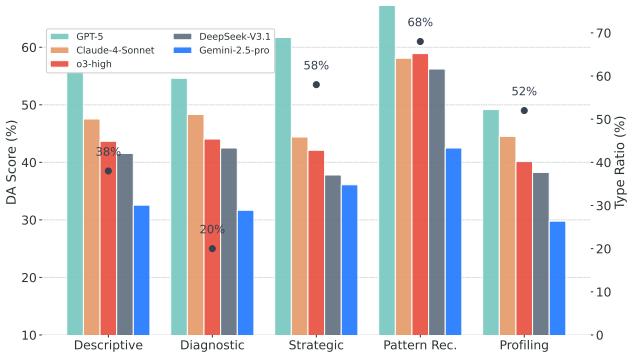

Figure 7: DA performance across five analytical objectives.

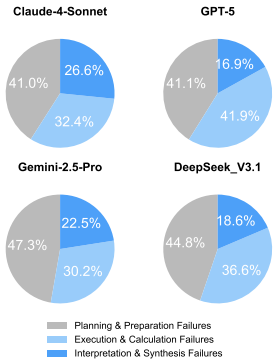

Figure 8: DA error distribution.

Error analysis. As shown in Fig. 8. To analyze DA failures, we classify errors into three stages: Planning (Tab. 11), Execution (Tab. 12), and Interpretation (Tab. 13). The dominant failure mode differs significantly across models. For instance, Gemini-2.5-Pro and DeepSeek-V3.1 are most prone to Planning failures, suggesting difficulty in understanding initial requirements. In contrast, GPT-5's primary weakness is in Execution, where its robust plans are undermined by flawed technical implementation. Claude-4-Sonnet shows a more balanced distribution of errors with no single bottleneck. For all models, Interpretation is the least common failure, indicating the primary challenges lie in either forming a correct plan or executing it flawlessly.

3.5 Validation of LLM-Judge Method

Systematic human-LLM agreement validation. To validate our LLM-as-Judge method, we conduct a systematic analysis across multiple levels of granularity. First, we collected responses from various models on the DA tasks and had our human annotators manually score each submission against all rubrics and GSB documents. We then established a reliable ground truth by measuring inter-rater agreement—that is, the degree of consistency among our human experts—which confirmed substantial consistency across all evaluation criteria (Tab. 4). With this human baseline, we benchmarked several candidate judges (e.g., Gemini 2.5 Flash, O4-Mini, GPT-4.1) at two primary levels of agreement:

(i) *case-level agreement*, which measures how consistently the judge scores a single task compared to human experts; and (ii) *model-level agreement*, which validates whether the judge's final ranking of all models matches the human-derived leaderboard. As shown in Tab. 4, across all levels of this comparison, both Gemini-2.5-Flash and O4-Mini demonstrate high and consistent alignment with human judgments. Given that O4-Mini's cost is ten times that of the former, we select Gemini 2.5 Flash as our standard judge model for its optimal balance of performance and cost-effectiveness. Furthermore, the Pearson correlation indicates that our LLM judge achieves a higher correlation with the human consensus than the average inter-rater correlation among humans, validating the soundness of our method.

Table 4: Comprehensive inter-rater agreement for DA tasks.

Metric	Rubric	GSB
Evaluation scale summary		
# Examples	50	50
# Cases	1120	250
Case-level (Pearson's r for Rubric, weighted κ for GSB)		
Inter-Rater	80.4	65.0
o4-mini	79.7	78.6
GPT-4.1	68.4	69.6
Gemini 2.5 Flash	79.4	75.9
Model-level (Pearson's r)	**Overall Score**	
o4-mini	97.9	
GPT-4.1	98.2	
Gemini 2.5 Flash	99.6	

Benchmarking stability. We assess the end-to-end stability of our evaluation pipeline. We quantify the total variability, arising from both the agent's stochasticity and the judge's grading, by repeating the entire evaluation 8 times. The results in Tab. 5 show that the standard deviations of the final scores are consistently small, demonstrating that our evaluation protocol yields statistically stable and replicable results.

Table 5: Variability of sub-task scores across 8 runs. Scores are percentages (mean $\pm$ std).

Model	DE-Arch	DA
GPT-5	62.5 ± 0.18	56.7 ± 0.16
DeepSeek-V3.1	51.2 ± 0.25	41.1 ± 0.22
Gemini 2.5 Pro	45.3 ± 0.21	32.1 ± 0.22
Claude-4 Sonnet	35.8 ± 0.19	43.8 ± 0.20
Qwen3-Max	18.1 ± 0.31	40.5 ± 0.29

4 Related Work

Agentic benchmarks. As LLM-based agents mature, benchmarks span tool use (Yao et al., 2024), software engineering (Jimenez et al., 2023; Zan et al., 2025), mobile interaction (Rawles et al., 2024), web navigation (Deng et al., 2023; Zhou et al., 2024), computer use (Xie et al., 2024), scientific discovery (Chen et al., 2024), and deep research (Phan et al., 2025; Wei et al., 2025), collectively advancing the field. In parallel, evaluation has moved beyond fixed-answer grading toward open-ended assessment (Li et al., 2024a; Wu et al., 2025; Du et al., 2025; Arora et al., 2025). DAComp is, to our knowledge, the first benchmark to cover the data-intelligence workflow, evaluating end-to-end data agents on both repository-level data engineering and open-ended data analysis, with the aim of advancing autonomous engineering and analytical capability.

Benchmarks for Data Agents. A data agent is an LLM-driven autonomous system that plans and executes end-to-end workflows, acquiring, transforming, and analyzing data via tool use and code execution to achieve user-defined objectives. Early work emphasizes single-shot tasks such as text-to-SQL (Yu et al., 2018; Li et al., 2024b) and code generation (Lai et al., 2023; Yin et al., 2023); more recent efforts push toward realistic SQL generation over real scenarios (Lei et al., 2024; Li et al., 2025), multi-turn data-science code generation (Hu et al., 2024; Huang et al., 2024; Jing et al., 2024) with iterative execution, and data analysis in business settings (Gu et al., 2024; Egg et al., 2025; Lai et al., 2025). DAComp goes beyond these efforts by introducing the first benchmark spanning enterprise data-intelligence workflows, encompassing repository-level engineering and open-ended analysis, and offering a rigorous testbed for advancing autonomous agents.

5 Conclusion

We introduce **DAComp**, a benchmark that uniquely models real-world enterprise data intelligence. It establishes the first testbed for repository-level data engineering, posing a formidable challenge to the code generation field. Simultaneously, it introduces the first evaluation framework for open-ended data analysis, guiding development beyond technical accuracy toward human-centric, actionable insights. Our experiments confirm its difficulty: even top agents post success rates below 20% on DE tasks and average scores under 40% on DA tasks. By providing this rigorous, holistic challenge, DAComp aims to steer research beyond isolated code generation and drive the evolution of truly capable autonomous data agents for the enterprise.

REFERENCES

Anthropic. Introducing Claude 4. https://www.anthropic.com/news/claude-4, 2025.

AI Anthropic. The claude 3 model family: Opus, sonnet, haiku. *Claude-3 Model Card*, 1:1, 2024.

Rahul K Arora, Jason Wei, Rebecca Soskin Hicks, Preston Bowman, Joaquin Quiñonero-Candela, Foivos Tsimpourlas, Michael Sharman, Meghan Shah, Andrea Vallone, Alex Beutel, et al. Healthbench: Evaluating large language models towards improved human health. *arXiv preprint arXiv:2505.08775*, 2025.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, et al. Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery. In *The Thirteenth International Conference on Learning Representations*, 2024.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023.

Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. Deepresearch bench: A comprehensive benchmark for deep research agents. *arXiv preprint arXiv:2506.11763*, 2025.

Alex Egg, Martin Iglesias Goyanes, Friso Kingma, Andreu Mora, Leandro von Werra, and Thomas Wolf. Dabstep: Data agent benchmark for multi-step reasoning. *arXiv preprint arXiv:2506.23719*, 2025.

Gemini. Gemini 2.5: Our most intelligent AI model. https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/, 2025.

Ken Gu, Ruoxi Shang, Ruien Jiang, Keying Kuang, Richard-John Lin, Donghe Lyu, Yue Mao, Youran Pan, Teng Wu, Jiaqian Yu, et al. Blade: Benchmarking language model agents for data-driven science. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 13936–13971, 2024.

Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, et al. Infiagent-dabench: Evaluating agents on data analysis tasks. In *Forty-first International Conference on Machine Learning*, 2024.

Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, et al. 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*, pp. 13487–13521, 2024.

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*, 2023.

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 to becoming data science experts?, 2024. URL https://arxiv.org/abs/2409.07703.

Eugenie Lai, Gerardo Vitagliano, Ziyu Zhang, Sivaprasad Sudhir, Om Chabra, Anna Zeng, Anton A Zabreyko, Chenning Li, Ferdi Kossmann, Jialin Ding, et al. Kramabench: A benchmark for ai systems on data-to-insight pipelines over data lakes. *arXiv preprint arXiv:2506.06541*, 2025.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pp. 18319–18345. PMLR, 2023.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, et al. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. *arXiv preprint arXiv:2411.07763*, 2024.

Haitao Li, Qian Dong, Junjie Chen, Huixue Su, Yujia Zhou, Qingyao Ai, Ziyi Ye, and Yiqun Liu. Llms-as-judges: a comprehensive survey on llm-based evaluation methods. *arXiv preprint arXiv:2412.05579*, 2024a.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36, 2024b.

Jinyang Li, Xiaolong Li, Ge Qu, Per Jacobsson, Bowen Qin, Binyuan Hui, Shuzheng Si, Nan Huo, Xiaohan Xu, Yue Zhang, et al. Swe-sql: Illuminating llm pathways to solve user sql issues in real-world applications. *arXiv preprint arXiv:2506.18951*, 2025.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

OpenAI. OpenAI GPT5 System Card. `https://cdn.openai.com/gpt-5-system-card.pdf`, 2025.

R OpenAI. Gpt-4 technical report. arxiv 2303.08774. *View in Article*, 2:13, 2023.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025.

Yuning Wu, Jiahao Mei, Ming Yan, Chenliang Li, Shaopeng Lai, Yuran Ren, Zijia Wang, Ji Zhang, Mengyue Wu, Qin Jin, et al. Writingbench: A comprehensive benchmark for generative writing. *arXiv preprint arXiv:2503.05244*, 2025.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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*, 2022.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. tau-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Pengcheng Yin, Wen-Ding Li, Kefan Xiao, Abhishek Rao, Yeming Wen, Kensen Shi, Joshua Howland, Paige Bailey, Michele Catasta, Henryk Michalewski, et al. Natural language to code generation in interactive data science notebooks. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 126–173, 2023.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 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*, pp. 3911–3921, 2018.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Linhao Zhang, Shulin Xin, Lu Chen, Qi Liu, Xiaojian Zhong, Aoyan Li, et al. Multi-swe-bench: A multilingual benchmark for issue resolving. *arXiv preprint arXiv:2504.02605*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024. URL https://arxiv.org/abs/2307.13854.

A Evaluation Methods Details

A.1 DAComp-DE-Impl/Evol

The DAComp-DE-Impl/Evol evaluated using three execution-based metrics that progressively increase in strictness: Component Score (CS), Cascading Failure Score (CFS), and Success Rate (SR). Fig. 9 illustrates how these metrics differ in scoring a simple pipeline when an intermediate node fails.

Component score (CS). Let $\mathcal{D}$ be the set of tasks. For task $d \in \mathcal{D}$, let layers be $\mathcal{L}$ (e.g., staging/intermediate/marts), and for each layer $\ell \in \mathcal{L}$ let $\mathcal{T}_{d,\ell}$ be its tables with weights $w_{d,t} \geq 0$. Define a table match indicator $m_{d,t} \in \{0,1\}$ by exact equivalence of *schema+data* between predicted and gold outputs (checked in DuckDB) under *perfect upstream inputs* (progressive/hybrid evaluation). The per-layer score and task-level CS are

$$S_{d,\ell} \;=\; \frac{\sum_{t\in\mathcal{T}_{d,\ell}} w_{d,t}\, m_{d,t}}{\sum_{t\in\mathcal{T}_{d,\ell}} w_{d,t}}, \qquad \mathrm{CS}_d \;=\; 100\cdot\sum_{\ell\in\mathcal{L}} \alpha_\ell\, S_{d,\ell}, \quad \text{with } \alpha_\ell \geq 0,\ \sum_{\ell} \alpha_\ell = 1.$$

We report the benchmark CS as $\mathrm{CS} = \frac{1}{|\mathcal{D}|}\sum_{d\in\mathcal{D}} \mathrm{CS}_d$.

Cascading failure score (CFS). For task d, let the pipeline DAG be $G_d = (V_d, E_d)$ with node weights $w_{d,j} \geq 0$ and ancestor set $\mathrm{Anc}_d(j)$. Let $m_{d,j} \in \{0,1\}$ be the node-level exact match (schema+data) under *predicted* upstreams. Define the cascading indicator recursively

$$s^{\mathrm{CFS}}_{d,j} \;=\; m_{d,j} \prod_{k\in\mathrm{Anc}_d(j)} s^{\mathrm{CFS}}_{d,k},$$

and the task-level CFS

$$\mathrm{CFS}_d \;=\; 100\cdot \frac{\sum_{j\in V_d} w_{d,j}\, s^{\mathrm{CFS}}_{d,j}}{\sum_{j\in V_d} w_{d,j}}.$$

We report $\mathrm{CFS} = \frac{1}{|\mathcal{D}|}\sum_{d\in\mathcal{D}} \mathrm{CFS}_d$.

Success rate (SR). A task is successful only if *every* component matches:

$$\mathrm{SR}_d \;=\; \prod_{j\in V_d} m_{d,j} \;\in\; \{0,1\}.$$

The benchmark success rate is the fraction of perfectly solved tasks:

$$\mathrm{SR} \;=\; \frac{1}{|\mathcal{D}|}\sum_{d\in\mathcal{D}} \mathrm{SR}_d.$$

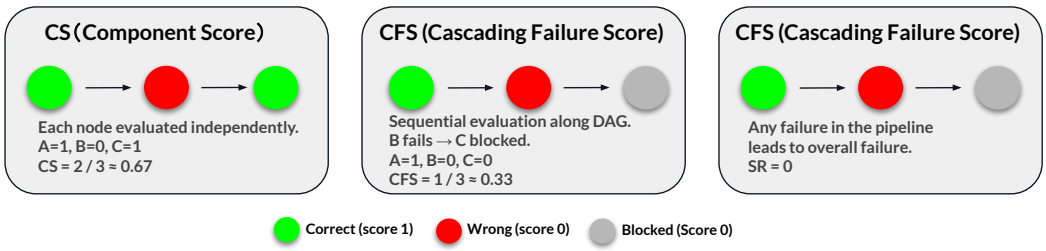

Figure 9: Illustration of how CS, CFS, and SR differ in scoring a simple pipeline when an intermediate node fails.

A.2 DACOMP-DE-ARCH

Three rubric dimensions. The evaluation of the DE-Arch tasks is conducted across three key dimensions, which are defined as follows:

1) Business Alignment and Semantic Accuracy: This dimension assesses how well the solution aligns with business requirements and ensures semantic correctness. It evaluates whether the proposed solution comprehensively addresses the task's objectives while maintaining semantic integrity in the context of the recruitment cost analysis system.

2) Technical Feasibility and Structural Completeness: This dimension evaluates the technical feasibility of the solution and the completeness of its structure. It checks whether the proposed model can be implemented successfully given the available resources and dependencies, and whether it adheres to necessary technical standards and best practices.

3) Design Quality: This dimension evaluates the design and clarity of the model. It looks at how well the model is structured, the clarity of its naming conventions, and the organization of the components. It also considers the use of modular design principles to ensure that the solution is maintainable and scalable.

DAComp-de-arch judge prompt. This prompt standardizes how a model blueprint is evaluated against a given user question and rubric. It defines clear scoring logic (deterministic vs. path-based criteria), enforces an evidence-first policy (no evidence, no points), and constrains the final score to requirement-level sums. A canonical JSON output schema captures per-criterion analysis, evidence, and scores, enabling reproducible, auditable assessments across tasks.

DE-Arch Judge Prompt

```
## Task Description
You are a professional data architect. Evaluate a model blueprint using the provided user
question and scoring rubric. First, study the rubric, then assess the blueprint strictly
according to the rubric and determine the extent to which it meets the standards.

## Scoring Framework
Total Score is the sum of all requirement scores. Each requirement contains multiple
scoring criteria:
1. Deterministic criteria: can be scored directly without considering different
implementation paths.
2. Non-deterministic criteria: may have multiple implementation paths. Select the best
matching path based on the assistants response and score using the sub-criteria of that
path. If no path clearly matches, use your own expertise to judge whether the response
satisfies the requirement goal. If it does, assign points, but the score for this
requirement cannot exceed the maximum of the defined paths.

## Final Scoring Logic
Final Score = sum of all requirement scores.
Requirement Score = sum of its criteria scores.
Each criterion score is one of: direct score, best matching path score, unmatched path
score, or sum of sub-criteria.

## Evidence Policy
Provide explicit evidence for every scored item. If evidence is missing, assign zero. If
uncertain, do not guess; assign zero.

<User Question Start>
{user_query}
</User Question End>

<Model Blueprint Start>
{model_blueprint}
</Model Blueprint End>

<Scoring Rubric Start>
{rubric}
</Scoring Rubric End>

You must analyze and score each rubric item one by one.

Response format:
{
  "Requirement1": {
```

```
    "Criterion1.1": {
      "Analysis": "Carefully read the content of the model blueprint, determine whether it
meets Criterion1.1, and assign a score",
      "Criterion1.1.x.1": {
        "Analysis": "Carefully read the content of the model blueprint, determine whether
it meets Criterion1.1.x.1, and assign a score",
        "Evidence": [],
        "Score": 0
      },
      "Criterion1.1.x.2": {
        "Analysis": "Carefully read the content of the model blueprint, determine whether
it meets Criterion1.1.x.2, and assign a score",
        "Evidence": [],
        "Score": 0
      },
      "Score": 0
    },
    "Criterion1.2": {
      "Analysis": "Analyze the reason for the best matching path, determine the best
matching path: Path1.2.x",
      "Criterion1.2.x.1": {
        "Analysis": "Carefully read the content of the model blueprint, determine whether
it meets Criterion1.2.x.1, and assign a score",
        "Evidence": [],
        "Score": 0
      },
      "Criterion1.2.x.2": {
        "Analysis": "Carefully read the content of the model blueprint, determine whether
it meets Criterion1.2.x.2, and assign a score",
        "Evidence": [],
        "Score": 0
      },
      "Score": 0
    },
    "Total Score": 0
  },
  "Requirement2": {
    "Criterion2.1": {
      "Analysis": "Analyze the reason for the best matching path, determine that there is
no best matching path. Based on your own knowledge, determine whether it meets Criterion2
.1. Referencing other paths, it should meet Criterion2.1.notfound.1: xxx; Criterion2.1.
notfound.2: xxx",
      "Criterion2.1.x.1": {
        "Analysis": "Carefully read the content of the model blueprint, determine whether
it meets Criterion2.1.x.1, and assign a score",
        "Evidence": [],
        "Score": 0
      },
      "Criterion2.1.x.2": {
        "Analysis": "Carefully read the content of the model blueprint, determine whether
it meets Criterion2.1.x.2, and assign a score",
        "Evidence": [],
        "Score": 0
      },
      "Score": 0
    }
  },
  "Total Score": 0
}
```


A.3 DACOMP-DA

A.3.1 HIERARCHICAL RUBRIC

Five rubric dimensions. The evaluation of DA tasks is conducted across five key dimensions, which are defined as follows:

***1) Completeness*:** This dimension assesses whether the agent's response comprehensively addresses all explicit and implicit requirements of the prompt. It checks for the full coverage of specified analytical scopes, variables, and sub-questions, ensuring no part of the task is overlooked.
***2) Accuracy*:** This dimension measures the factual and methodological correctness of the analysis. It includes the validity of the code logic, the correctness of calculations, and the factual precision of all reported figures and statistical results against a verifiable ground truth.
***3) Insightfulness*:** This dimension evaluates the agent's ability to move beyond mere data reporting

to generate valuable interpretations. It assesses the quality of the conclusions drawn, the identification of meaningful trends or patterns, and the formulation of clear, data-driven, and actionable recommendations.

***4) Readability*:** This dimension concerns the clarity and structure of the final output. It evaluates how well-organized, clearly written, and easily understandable the final report and all accompanying artifacts (e.g., code, tables, visualizations) are for a human audience.

***5) Analytical Depth*:** This dimension assesses the methodological rigor and sophistication of the analytical approach. It distinguishes between superficial analyses (e.g., simple averages) and more profound approaches that involve appropriate statistical tests, control for variables, and demonstrate a deeper understanding of the underlying data and business context.

Hierarchical rubrics example. As shown in Tab. 6, we provide a hierarchical scoring rubric that decomposes the task into requirements and sub-standards, with explicit checkpoints and point allocations for consistent evaluation.

Hierarchical rubric prompt. Below is the Hierarchical Rubric Prompt.

Hierarchical Rubric Prompt

```
# Task Description
You are a data analysis expert. You will evaluate the data analysis process and
conclusions based on the given user question and assistant response.
Your task is to read a data-analysis-oriented response and a scoring rubric, and then,
according to that rubric, determine how well the assistant's response meets the criterion.

# Assistant Response
<<assistant_response>>

# Scoring Criterion
<<rubric>>

# Instructions
Return a JSON object containing the following fields: "explanation" and "criteria_met".
- "explanation" should be a string explaining why the response does or does not satisfy
the criterion.
- "criteria_met" should be a boolean (true/false) indicating whether the response
satisfies the criterion. If a criterion contains multiple sentences or sub-criteria, all
must be considered. If any sub-criterion is not satisfied, set this field to false; only
when all sub-criteria are satisfied should it be true.

# Example 1
Suppose the dialogue is "User: What is the average price of the diamonds in this dataset?
Assistant: The average price of the diamonds is $5000.", and after calculation, the
correct average is about $3932. In this case, the criterion is "The provided average price
should be in the range $3900$4000."

'''json
{
  "explanation": "The assistant provided $5000, which is not within the required
$3900$4000 range, so it does not meet the criterion.",
  "criteria_met": false
}
'''

# Example 2
Please note that sometimes the rubric may describe an undesirable condition with a
negative score. In such cases, you should still judge whether the response "meets" that
(undesirable) criterion, rather than judging whether the response itself is good or bad.

Case A (High-quality response):
Dialogue:
"User: Compute the energy consumption share of each department. Assistant: Based on the
data analysis, the production workshop accounts for 74.99% of total energy consumption,
the R&D center 12.51%, warehousing and logistics 11.57%, and the office area 0.92%. The
calculation process is complete and data sources are clear."
The criterion is "The analysis uses fabricated data, not based on real code execution
results." Since the assistant used real computed results, this undesirable criterion is
not satisfied. Return:

'''json
{
```

Table 6: Hierarchical rubric for the business analysis task defined as follows: *Compare the business performance across the four major regions (Central, East, South, West), analyze the differences in penetration rate and profitability of each region in the three market segments (Consumer, Corporate, Home Office) during 2015, 2016, and 2017, identify the region-market combination with the best performance, and provide recommendations for expansion.*

Requirement & Standard		Path	Item (Sub-standard) & Key Description	Points
Req. 1: Penetration & Profitability Analysis (Max 8 pts)	**Std. 1.1:** Penetration Rate Analysis	**1.1.A** (Sales)	**1.1.A.1 (Completeness):** Define & calculate sales penetration (annual + 3-yr avg).	1
			1.1.A.2 (Accuracy): Calculations must match anchors (e.g., West-Consumer avg $\approx 29.72\%$).	2
			1.1.A.3 (Conclusion): Derive $\geq$3 valid conclusions on market position (e.g., East-/West duopoly).	1
		1.2.B (Risk-Adj. Margin)	**1.2.B.1 (Completeness):** Define & calculate risk-adjusted profit margin (e.g., mean $- 0.5 \times$ std).	1
			1.2.B.2 (Accuracy): Calculations must match anchors (e.g., Central-Home Office adj ≈ 16.37).	2
			1.2.B.3 (Conclusion): Derive $\geq$2 insights on risk/return (e.g., identify stable vs. high-risk yields).	1
	Std. 1.2: Profitability Analysis	**1.2.A** (Basic Margin)	**1.2.A.1 (Completeness):** Define & calculate basic profit margin (annual + 3-yr avg).	1
			1.2.A.2 (Accuracy): Calculations must match anchors (e.g., Central-Corporate $\approx$ 20.22%).	1
			1.2.A.3 (Conclusion): Derive $\geq$2 conclusions on profit tiers and strategic priorities.	1
		1.1.B (Orders)	**1.1.B.1 (Completeness):** Define & calculate order penetration (annual + 3-yr avg).	1
			1.1.B.2 (Accuracy): Cross-validate sales vs. order trends; calculations must be correct.	1
			1.1.B.3 (Conclusion): Analyze avg. order value to derive insights on customer structure.	1
Req. 2: Regional Perf. Comparison (Max 3 pts)	**Std. 2.1:** Multi-dim. Evaluation	**2.1.A** (Weighted Score)	**2.1.A.1 (Completeness):** Define & compute a weighted composite score from normalized penetration profit.	
			2.1.A.2 (Accuracy): Final rankings are consistent with the chosen weights and normalized values.	1
			2.1.A.3 (Conclusion): Derive regional roles (Leaders, Potentials, etc.) based on composite scores.	1
Req. 3: Identify Best Combo (Max 2 pts)	**Std. 3.1:** Optimal ID	**3.1.A** (Composite Rank)	**3.1.A.1 (Accuracy):** Identify TOP3 combinations using a weighted score; must match $\geq$2 anchors (e.g., East-Home Office pen. $\approx 35.00\%$, margin $\approx 18.06\%$).	1
			3.1.A.2 (Conclusion): Analyze TOP3 for strategic value (core vs. growth) and inherent risks.	1
Req. 4: Expansion Strategy (Max 2 pts)	**Std. 4.1:** Strategic Recs.	**4.1.A** (Action Plan)	**4.1.A.1 (Conclusion):** Provide a comprehensive plan covering: • Business positioning & priorities • Actionable steps with KPIs • Strategic rationale & risk control • Implementation timeline	2

```
  "explanation": "The assistant's data matches actual computation and clearly states the
process and data sources, with no fabricated data. Therefore, the (undesirable) criterion
is not satisfied.",
  "criteria_met": false
}
```

```
Case B (Low-quality response):
Dialogue:
"User: Analyze the relationship between the share of industrial water use and economic
development. Assistant: Generally, regions with a higher share of industrial water have
better economic development, with about 60% correlation. I think this trend is reasonable."
The criterion is "The analytical conclusion lacks a specific correlation coefficient
calculation and uses subjective speculation rather than data-driven conclusions." Since
the assistant did not provide a concrete calculation and used subjective judgment, this
undesirable criterion is satisfied. Return:
```

```json
{
  "explanation": "Statements such as 'about 60% correlation' and 'I think this trend is
reasonable' are subjective, without a specific correlation computation or data support.
Therefore, the (undesirable) criterion is satisfied.",
  "criteria_met": true
}
```

```
In other words, for negative-scoring criteria: a high-quality response should be judged as
false (because it does not meet the undesirable criterion), and only low-quality responses
that actually meet the undesirable criterion should be judged as true.

# Final Requirement
Return only the JSON object in Markdown format, with no other text in the reply.
```

A.3.2 GOOD-SAME-BAD JUDGE

Good-Same-Bad Judge Prompt

```
You are a data analysis evaluation expert. You need to judge whether the following two
reports are good or bad.
Evaluate them in detail from the following two dimensions:
1. The report is highly readable and easy to understand.
2. The analysis is professional and in-depth.

Give a score for each dimension, with a range of -10 to 10.
Notes:
+ The analysis and scoring are comparative: compare the report to be evaluated with the
baseline report.
+ -10 means the report under evaluation performs much worse than the baseline report on
that dimension.
+ 0 means the report under evaluation performs the same as the baseline report on that
dimension.
+ 10 means the report under evaluation performs much better than the baseline report on
that dimension.
+ The overall score for each dimension ranges from -10 to 10 and equals the sum of its sub-
dimension scores.

Details:
Readability is specifically reflected in the following sub-dimensions:
- Convey complex information concisely so readers can quickly grasp key points (e.g., use
Markdown to structure the report; use bold/italic to highlight key information). Score
range: -4 to 4.
- Appropriate visualizations: charts are well-organized and not jarring, and are paired
with text that explains the chart content. Score range: -3 to 3.
- Follows a clear writing structure, such as a "general--specific--general" flow, with
clear hierarchy (e.g., use subheadings). Score range: -2 to 2.
- Concise language: avoid verbosity and repeated expressions. Score range: -1 to 1.

Professionalism and depth of analysis are reflected in the following sub-dimensions:
- Analyze from multiple dimensions and perspectives, considering different factors and
scenarios. Score range: -4 to 4.
- Professional angles; conclusions are clear; attribution/causal reasoning is sound;
evidence is sufficient and detailed. Score range: -3 to 3.
- Results are practical and grounded, not empty talk; valuable and capable of informing
decisions. Score range: -2 to 2.
- Estimate the potential impact of recommendations. Score range: -1 to 1.
```

```
Output format:
```json
{
    "Readability": {
        "Analysis": "On sub-dimension xxx, the baseline report's strengths/weaknesses are
xxx, and the report under evaluation's strengths/weaknesses are xxx. Contrastive analysis
of the differences; the report under evaluation scores xx on this sub-dimension.",
        "Summary": "Summary of the readability analysis for the report under evaluation",
        "Score": int
    },
    "Analytical Depth": {
        "Analysis": "On sub-dimension xxx, the baseline report's strengths/weaknesses are
xxx, and the report under evaluation's strengths/weaknesses are xxx. Contrastive analysis
of the differences; the report under evaluation scores xx on this sub-dimension.",
        "Summary": "Summary of the professionalism and depth analysis for the report under
evaluation",
        "Score": int
    }
}
```
```

B EXPERIMENTS SETTING

B.1 AGENT BASELINE

For our data engineering baseline, we develop an agent framework inspired by the **ReAct** (Yao et al., 2022). This framework enables the agent to perform complex, repository-level tasks through multi-turn interactions within a sandboxed, interactive file system environment.

To facilitate these interactions, we define a concise yet powerful set of four actions, as detailed in Tab. 7. The agent iteratively generates a thought process, selects an action, and observes the outcome from the file system, continuing this loop until the task is complete. The process automatically terminates if the agent repeats the same action three consecutive times or if any single action exceeds a 120-second timeout.

Table 7: The core action space for our DE agent baseline. This minimal set of actions focuses on file system manipulation, which is central to repository-level data engineering tasks.

Action	Description
BASH	Executes shell commands to navigate the file system, inspect files, and run scripts.
CREATE_FILE	Creates a new file with specified content.
EDIT_FILE	Edits or overwrites the content of an existing file.
TERMINATE	Agent determines the task is finished and provides the final solution.

B.2 OPENHANDS DETAILS

We have integrated OpenHands(Wang et al., 2024) into our DE and DA tasks, utilizing the Codeact agent. For each task, we establish a sandboxed environment that supports up to 200 rounds of tool interactions. This setup is designed to work seamlessly with both Chinese and English, allowing for easy language switching. Three sets of tools are provided, as detailed in Tab. 8.

Table 8: The Core Action Space for OpenHands. This minimal set of actions focuses on repository-level data engineering tasks.

Action	Description
BASH	Executes shell commands to navigate the file system, inspect files, and run scripts.
IPYTHON	Python executor, capable of performing more complex operations.
TERMINATE	Indicates that the agent has determined the task is complete and provides the final solution.

Table 9: Detailed performance breakdown for various agent systems on DAComp-DA tasks.

Method	DA Score	Rubric Score	GSB Score	Completeness	Accuracy	Insightfulness	Readability	Analytical Depth
OpenHands Framework								
GPT-5	**52.06**	**52.97**	48.40	**68.53**	**49.91**	**57.75**	36.80	**60.00**
o3	48.78	46.95	56.10	61.75	44.83	52.46	66.40	45.80
Claude-4-Sonnet	51.33	48.87	**61.20**	66.37	44.62	56.35	63.00	59.40
Gemini-2.5-Pro	40.74	37.87	52.20	52.25	34.49	47.11	**67.20**	37.20
DeepSeek-V3.1	45.79	43.50	54.98	59.52	40.31	51.38	61.60	48.35
Qwen3-Coder	41.88	43.50	35.40	58.61	41.86	49.21	42.40	28.40
Qwen3-235B-A22B	21.29	24.47	8.60	34.15	22.45	27.65	11.60	5.60
Qwen3-8B	3.16	3.75	0.80	9.66	4.13	1.03	1.00	0.60
Agentic Baseline (Avg By ID)								
GPT-5	**56.79**	**55.90**	**60.36**	**69.85**	**51.80**	**64.87**	48.73	**72.00**
o3	44.13	44.53	42.53	58.11	40.69	53.66	43.48	41.59
Claude-4-Sonnet	43.89	43.94	43.68	57.17	39.95	49.06	56.88	30.49
DeepSeek-V3.1	41.05	41.38	39.75	53.06	37.62	46.05	50.03	29.48
Qwen3-235B-A22B	40.51	36.33	57.24	48.88	32.08	47.03	**70.30**	44.19
o4-mini	39.48	43.37	23.93	55.48	41.23	48.11	23.30	24.56
Moonshot-Kimi-K2-0905	35.66	34.38	40.77	49.61	29.93	44.16	47.63	33.91
Doubao-1.6-Thinking	35.19	33.76	40.93	48.33	30.17	42.06	55.82	26.05
Gemini-2.5-Pro	32.14	31.44	34.94	42.55	28.16	36.98	50.78	19.11
Gemini-2.5-Flash	31.56	33.37	24.30	45.17	29.98	39.23	30.53	18.08
DeepSeek-R1-250528	30.82	28.93	38.40	43.01	26.14	34.10	51.69	25.11
Qwen3-Coder	28.08	30.87	16.91	44.64	28.84	33.09	22.22	11.60
GPT-4.1-mini	21.61	26.45	2.24	37.93	25.81	26.76	1.18	3.31
Commercial Agent Systems								
ChatGPT-Agent	**45.90**	**41.95**	**61.70**	**52.06**	**44.67**	30.81	**58.00**	**65.40**
Claude-Agent	29.53	30.97	23.80	49.20	28.10	**34.07**	26.80	20.80

B.3 Additional Experimental Results

Task complexity and scale are key determinants of performance. The overall complexity of a data engineering task, measured by the number of nodes in the dependency graph or the total lines of code, strongly impacts agent performance, as shown in Fig. 10. For *Implementation* tasks, we observe a general decline in the Component Score as the number of nodes increases, with models like GPT-5 showing a significant performance drop on tasks with more than 50 nodes. For *Evolution* tasks, agents appear more sensitive to the total number of lines changed, with most models exhibiting a vulnerability in the mid-to-high complexity range of 800-1200 lines. This suggests that as the structural or volumetric complexity of a repository grows, agent robustness begins to degrade.

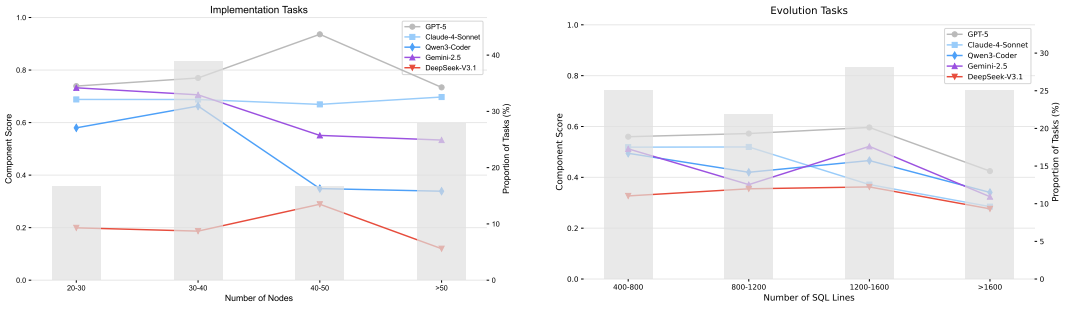

Figure 10: Effect of node count and line count.

C Examples

C.1 DE-Architecture Task

This task aims to derive a data engineering blueprint for a business question. As an illustration, we present a Salesforce-related question along with its evaluation rubrics.

DE-Architecture: Business Requirement

```
Can we build a "true performance profile" for each sales representative? I want to
understand not just their sales volume, but more importantly, the quality of the customers
they acquire. Will these customers continue to do business with us? And do details of the
```

```
sales process (like the pace of opportunity advancement, customer communication frequency,
etc.) affect the long-term value of the customer?
```

DE-Architecture: Evaluation Rubric

```
Requirement I: Business Alignment & Semantic Accuracy
- Ensures that the data models correctly reflect the core business logic.
- Customer Metrics:
  * Customer quality, LTV, and repeat business metrics must be correctly attributed.
  * Metrics must fall within logically valid ranges (e.g., 0--100).
- Sales Process Metrics:
  * Sales cycle and communication quality scores must be implemented and populated.
  * Metrics must demonstrate realistic values.

Requirement II: Technical & Structural Integrity
- Validates the technical soundness and completeness of the data tables.
- Model Completeness:
  * The final mart table (...performance_profile) must be fully populated for all valid
profiles.
  * No nulls allowed in key identifier fields.
- Data Consistency:
  * Records for each sales representative must be consistent across all related
intermediate and mart tables.
- Sufficient Volume:
  * The pipeline must produce at least 200 valid profiles to ensure analytical robustness.

Requirement III: Analytical Value & Logic
- Verifies that the final outputs provide meaningful insights and adhere to business
hypotheses.
- Value Profile Classification:
  * The "Tree Planter" classification for high-value reps must be applied to all eligible
candidates.
  * Must identify a sufficient cohort (e.g., >= 150).
- Business Logic Validation:
  * The final model must satisfy key business hypotheses.
  * Example: a positive correlation between customer quality scores and repeat business
rates.
```

C.2 DE-IMPLEMENTATION TASK

This task evaluates an agent's ability to build an entire data engineering repository from scratch based on a detailed technical specification.

DE-Implementation: DE Design Specifications

```
  staging_layer:
    example: stg_salesforce__account
    purpose: >
      Transform raw Salesforce account records into clean staging tables.
      Apply heavy-duty data cleaning:
      - normalize_email(), format_phone()
      - enforce DECIMAL(15,2) precision on revenue
      - quarantine() invalid records, nullify_field() for soft failures
      Guarantee: no null in account_id, owner_id; business fields standardized.
    ... ...
  intermediate_layer:
    example: int_salesforce__account_enhanced
    purpose: >
      Construct enriched account model with business logic.
      Join staging tables with user dimension     add owner + hierarchy info.
      Add derived fields (activity_score, account_health).
      Grain = "1 row per account".
      Note: Designed as reusable building block for multiple marts.
    ... ...
  marts_layer:
    example: fct_salesforce__sales_pipeline
    purpose: >
      Deliver pipeline fact table for exec-level analytics & forecasting.
      Row grain = "1 opportunity per reporting_date".
      Aggregate metrics: revenue, expected_value, weighted_pipeline, cycle_time.
      Attach dimensions: region, industry, owner, fiscal_calendar.
      Feeds dashboards, KPIs, and predictive modeling.
```

DE-Implementation: Ground-truth DE project repository

```
Staging Layer:
    stg_salesforce__account_history.sql, stg_salesforce__account.sql,
    stg_salesforce__contact_history.sql, stg_salesforce__contact.sql,
    stg_salesforce__event.sql, stg_salesforce__lead.sql,
    stg_salesforce__opportunity_history.sql, stg_salesforce__opportunity_line_item.sql,
    stg_salesforce__opportunity.sql, stg_salesforce__order.sql,
    stg_salesforce__product_2.sql, stg_salesforce__task.sql,
    stg_salesforce__user_role.sql, stg_salesforce__user.sql

Intermediate Layer:
    int_salesforce__account_enhanced.sql, int_salesforce__activity_summary.sql,
    int_salesforce__date_spine.sql, int_salesforce__lead_conversion_funnel.sql,
    int_salesforce__opportunity_aggregation_by_owner.sql,
    int_salesforce__opportunity_pipeline.sql, int_salesforce__user_performance.sql

Mart Layer:
    dim_salesforce__user.sql, fct_salesforce__account_engagement.sql,
    fct_salesforce__lead_performance.sql, fct_salesforce__sales_pipeline.sql,
    salesforce__account_daily_history.sql, salesforce__contact_daily_history.sql,
    salesforce__contact_enhanced.sql, salesforce__daily_activity.sql,
    salesforce__manager_performance.sql, salesforce__opportunity_daily_history.sql,
    salesforce__opportunity_enhanced.sql, salesforce__opportunity_line_item_enhanced.sql,
    salesforce__owner_performance.sql, salesforce__revenue_analytics.sql,
    salesforce__sales_snapshot.sql, salesforce__team_performance.sql
```

C.3 DE-EVOLUTION TASK

This task evaluates an agent's ability to plan, surface complete requirements, and produce SQL by adapting an existing SQL repository to a revised business specification—identifying scope and metric changes, updating definitions and dependencies, and delivering a final, fit-for-purpose project that fully aligns with the new requirement.

DE-Evolution: Requirement Specifications

```
Business Pain Point:
  - Current opportunity management lacks robust cost-effectiveness analysis.
  - Cannot measure acquisition cost, maintenance, and ROI consistently.

Objectives:
  - Multi-dimensional cost allocation (travel, marketing, labor, shared resources).
  - Lifecycle  c o s t revenue  matching (one-time, subscription, multi-year).
  - Multi-scenario ROI analysis with sensitivity & scenario modeling.

Implementation Highlights:
  - Flexible allocation rules (time weighting, channel path, dynamic labor rates).
  - ROI logic per revenue model (rolling 12M, discounted LTV, IRR).
  - Time-based alignment of costs and revenues.
  - Data quality checks (missing value fill, anomaly detection).
```

DE-Evolution: Ground-truth solution

```
Modified SQL:
  - int__opportunity_pipeline.sql
  - fct__sales_pipeline.sql
  - revenue_analytics.sql
  - fct__account_engagement.sql

Key Enhancements in fct__sales_pipeline:
  - Added cost allocation fields (acquisition, travel, marketing, labor).
  - Added ROI metrics (roi_percentage, cost_per_dollar_revenue, LTV ratio).
  - Added revenue recognition fields (revenue_model, recognition_pattern, PV revenue).
  - Added cost variance & risk indicators (variance %, anomaly flag, risk level).
  - Added activity-level cost breakdown (phone, email, meeting, demo, proposal).
  - Added efficiency & ranking metrics (cost_efficiency_tier, investment_priority_rank).
```

C.4 DA TASK

In this section, we show the detailed classification of the task types solved by DAComp-DA in Tab. 10.

Table 10: Definitions and Examples for the Five DA Task Type Categories.

Category Name	Definition & Objective	Example Question
Descriptive	Focuses on summarizing historical data to answer "What happened?". Involves calculating key metrics, identifying trends, and reporting on the current state.	*Analyze sales trends in the three categories of office supplies, technology, and furniture from 2015 to 2018, identify the fastest-growing product category for each year, and evaluate performance differences among regional managers based on regional sales data.*
Diagnostic	Aims to uncover the root causes of a particular outcome, answering "Why did it happen?". Involves drilling down into data, identifying anomalies, and discovering factors that influence a result.	*For the product category with the greatest annual volatility, investigate the underlying reasons. Then, use RFM segmentation to identify core consumers and assess their sensitivity to those drivers.*
Strategic	Focuses on providing data-driven recommendations for future actions, answering "What should we do?". It translates insights from descriptive and diagnostic analysis into concrete, actionable plans.	*As the sales leader for Coca-Cola, which sales outlet types should I increase or decrease our contracts with? Please provide recommendations based on an analysis of key data such as sales target attainment, customer complaints, and sales volume.*
Pattern Recognition	Involves exploring data to uncover previously unknown relationships, correlations, or patterns, answering "What are the hidden connections?". It is often open-ended and seeks to generate new hypotheses.	*Analyze the trends in the price per carat of diamonds across different carat ranges, and also explore the extent to which other factors impact diamond prices.*
Profiling	Aims to group a population (e.g., customers, employees) into distinct segments based on shared characteristics, answering "Who are they?". The goal is to understand the composition and behavior of different groups.	*Based on a comprehensive ranking that considers effective work hours, overall production quantity, and quality, please analyze the characteristics of our top performers and recommend the ideal profile for future hires.*

D ERROR ANALYSIS

D.1 DE-ARCHITECTURE CASE

As shown in Fig. 11, we present a "DE-Arch Error Case" panel: the left side shows a minimal blueprint, while the right side scores 16 checklist items (final score: 5/16), revealing several systemic weaknesses.

D.2 DE-IMPLEMENTATION CASE

It is crucial to prevent implementation issues—such as improper joins, flawed aggregations, and circular dependencies. The cases in Fig. 12 and Fig. 13 serve as representative DE-Impl examples.

DE-Arch Error Case

We've noticed in recent months that a number of high-value customers have ceased all system activity within 30-90 days of converting to paid accounts, seemingly churning. Strangely, these customers appear highly engaged in Marketo and are actively paying on Stripe, yet Zendesk shows virtually no support tickets. I need help building a comprehensive "silent customer risk warning system" that can identify these seemingly healthy but potentially problematic customers and predict their potential churn risk by analyzing behavioral changes across systems.

```
# ===== Incorrect (Minimal) Blueprint =====
modeling_spec:
  purpose: "Detect silent customers post-conversion."
  design_principles:
    grain_importance: "Declared but not enforced"
    time_windows: "pre-30d, post-31-60d"
  intermediate_models:
    - name: int_marketo
      source_models:
        - raw.marketo_leads
        - stg_customer360__nicknames
      business_logic: "Normalize emails only"
      columns:
        - name: lead_id
    - name: int_stripe
      source_models:
        - raw.stripe_charges
      business_logic: "Aggregate payments"
    - name: int_matches
      source_models:
        - int_marketo
        - int_stripe
        - int_zendesk
      business_logic: "Match by email/phone"
  marts_models:
    - name: mart_silent_risk
      source_models:
        - int_matches
      business_logic: "Compute risk_score"
      columns:
        - name: customer360_id
        - name: risk_score
assumptions:
  notes: "Exports only"
```

5/16

- ✅ Clear business goal (1/1)
- ❌ Not enforced downstream (0/1)
- ❌ Ambiguous boundaries / no timezone (0/1)
- ✅ Valid source (1/1)
- ❌ Dangling reference (0/1)
- ❌ Missing fallback/edge-case handling (0/1)
- ❌ No tests (not_null/unique) (0/1)
- ✅ Valid source (1/1)
- ❌ No refund/multi-currency handling (0/1)
- ❌ Dangling reference (0/1)
- ❌ No thresholds/weights/formulas (0/1)
- ✅ Proper mart structure (1/1)
- ❌ No formula, weights, or normalization (0/1)
- ✅ ID column defined (1/1)
- ❌ Metric placeholder only (0/1)
- ❌ Missing refresh frequency / freshness (0/1)

Figure 11: DE-Arch error case. Key issues include: (1) a clear business objective but weak downstream enforcement, ambiguous boundaries, and no timezone convention; (2) dangling references in intermediate models, missing fallback and edge-case handling, and absence of basic tests such as not_null/unique; (3) no treatment for refunds and multi-currency scenarios; (4) missing thresholds, weights, and formulas in aggregation and metric layers, with some fields not provided by sources; (5) placeholder metrics only and no refresh-frequency/freshness policy. Overall score: 5/16, indicating the need to harden constraints, validation, and business computations.

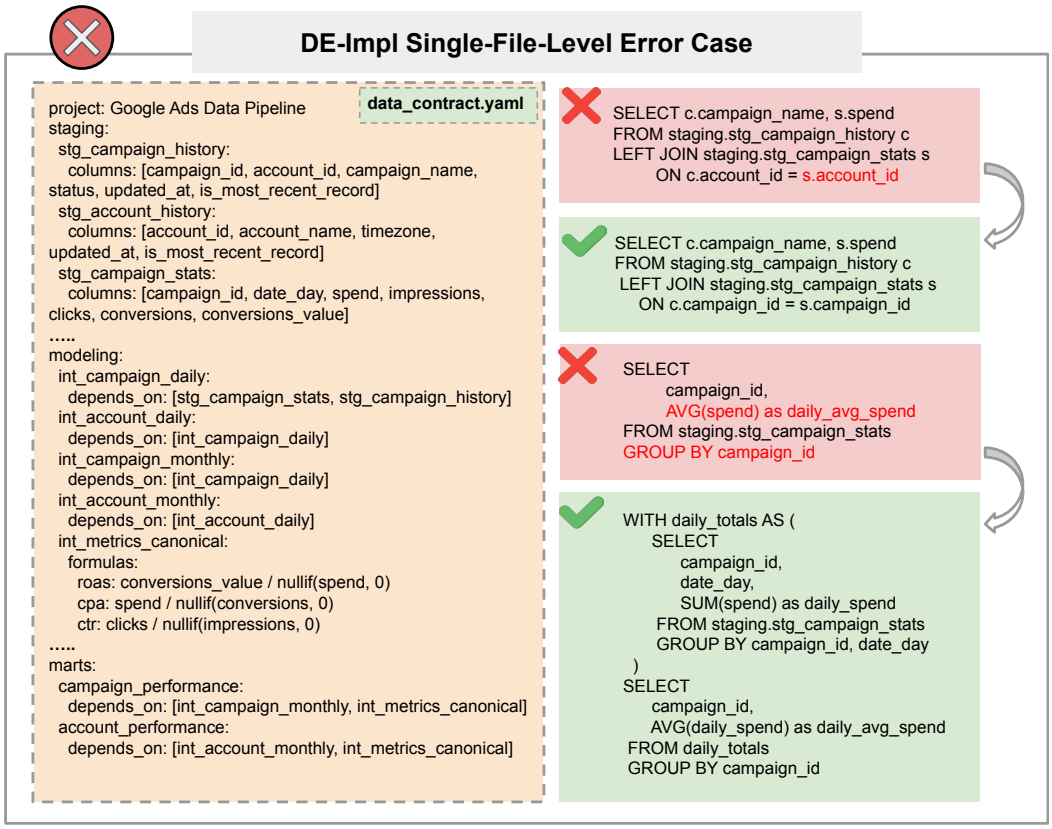

Figure 12: Examples of errors in DE-Impl: the red-crossed cases show mistakes such as joining on mismatched keys (`account_id` instead of `campaign_id`) and incorrect aggregation without respecting daily granularity, while the green-checked cases illustrate valid implementations with proper joins and staged aggregation.

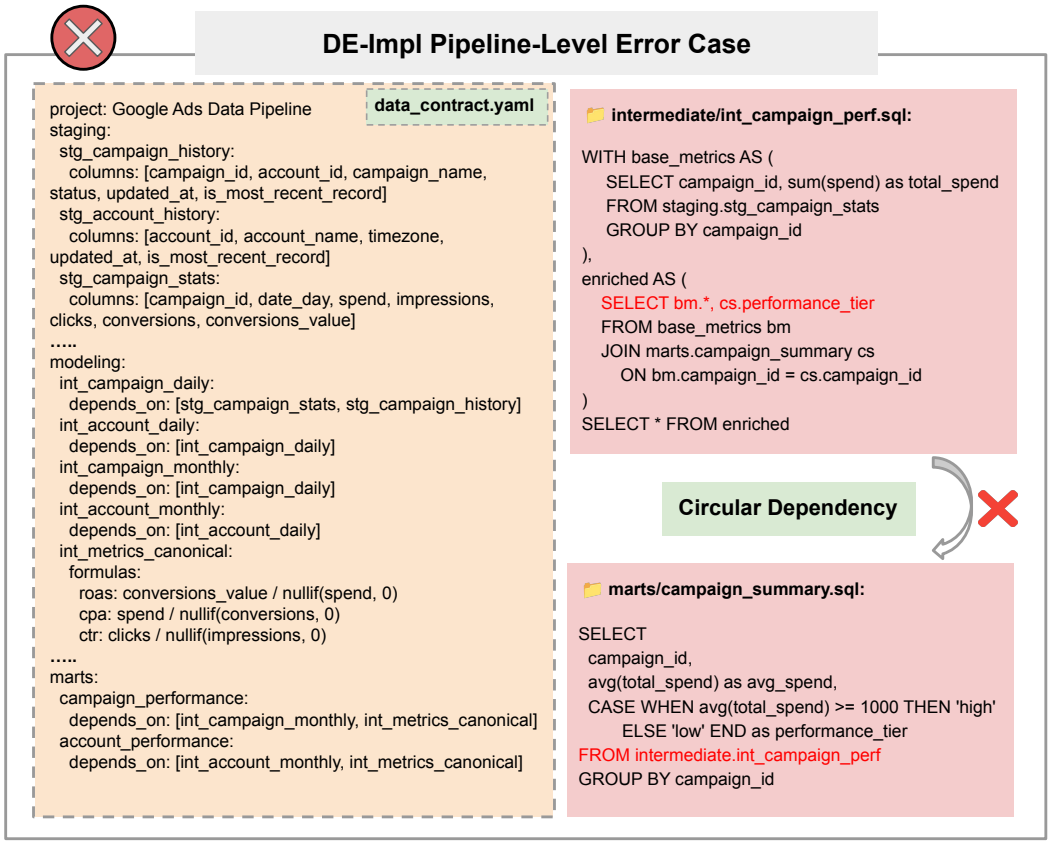

Figure 13: Examples of errors in DE-Impl: a circular dependency in which `int_campaign_perf.sql` depends on `campaign_summary.sql`, creating a loop in the data pipeline.

D.3 DE-EVOLUTION CASE

To illustrate how evolution errors can propagate across layers and distort downstream business metrics, we present a pipeline-level DE-Evol example in Fig. 14.

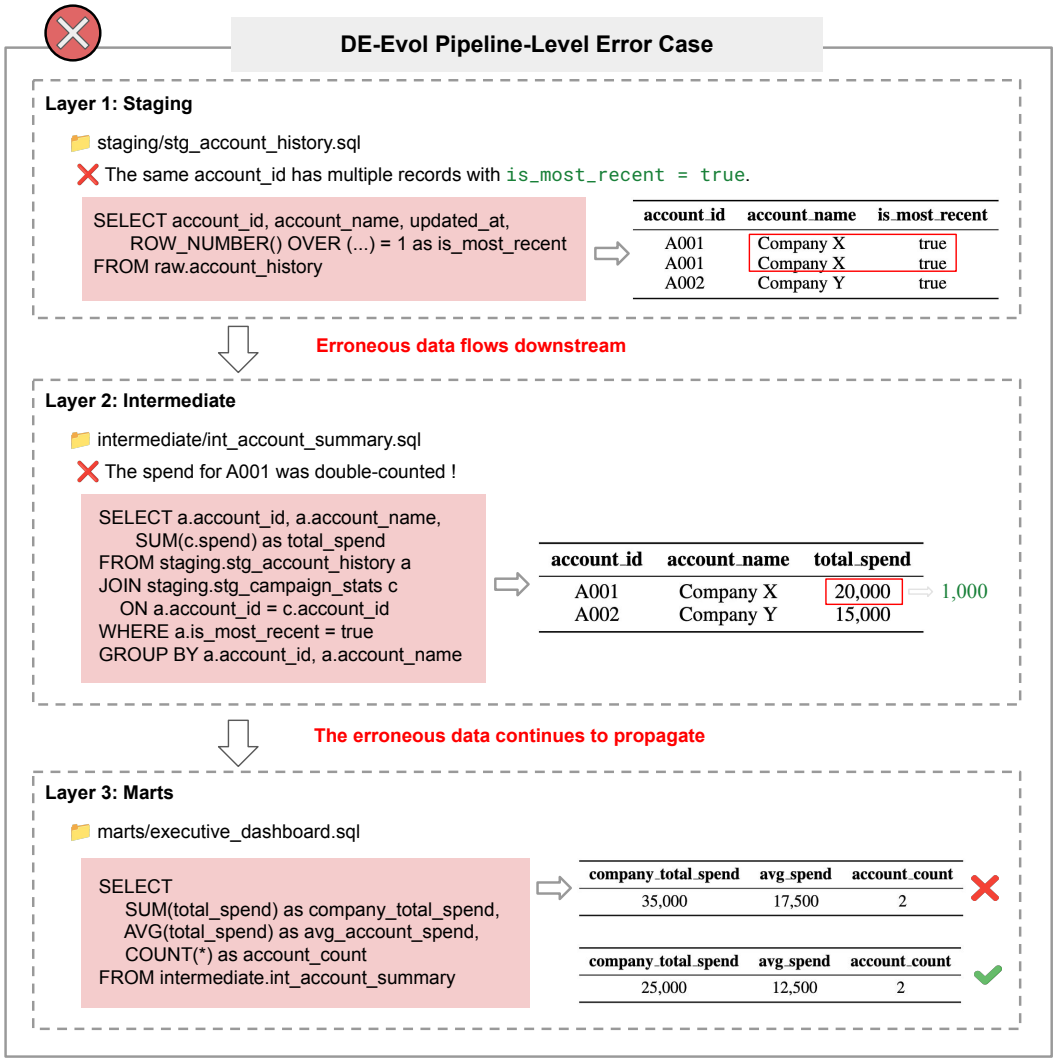

Figure 14: DE-Evol pipeline-level error case. Layer 1 (Staging) contains duplicate "current" rows where the same `account_id` appears multiple times with `is_most_recent = true`. Layer 2 (Intermediate) joins to campaign stats while filtering on `is_most_recent = true`, causing A001's spend to be double-counted (`total_spend` becomes 20,000 instead of 10,000). Layer 3 (Marts) aggregates the erroneous intermediate table, inflating `company_total_spend` and `avg_account_spend` (35,000 and 17,500) compared with the correct values (25,000 and 12,500). The figure highlights how a seemingly small staging inconsistency can cascade into materially incorrect executive metrics.

D.4 DA CASE

To illustrate typical failure modes in DA tasks, Tab. 11, Tab. 12, and Tab. 13 present focused case studies of Planning, Execution, and Interpretation errors, respectively. First, a scoping lapse omitted required unstructured data, yielding a biased sample and invalidating all downstream analysis. Second, despite a sound plan, a key metric was computed with an incorrect formula (simple average instead of weighted average), producing misleading channel insights. Third, even with flawless

calculations, the agent failed to synthesize findings into a context-aware conclusion and omitted mandatory limitations and a safety disclaimer. Together, these cases demonstrate that reliable DA outputs require aligned rigor across *planning, implementation, and interpretation*, with checks that prevent any single stage from compromising the whole.

Table 11: Focused case study of a critical Planning Error. This table analyzes the agent's plan against a pivotal standard (Data Scoping), highlighting omitted steps (in red) that led to a fundamentally flawed analysis.

Required Planning Step (from Rubric)	**Agent's Plan vs. Actual Action**	**Outcome**
Case Study: Standard 1.1 — Data Understanding & Scoping		
Step 1.1.A.1: Filter using the structured `Education Requirement` column.	The agent correctly planned and executed this step.	✓ **PASS**
Step 1.1.A.2: Additionally extract candidates from the `Job Description` column.	**CRITICAL PLANNING FAILURE:** This step was entirely omitted from the agent's plan; it never considered searching this column.	✗ **FAIL**
Step 1.1.A.3: Further apply complex filtering rules to the `Job Description` column.	**CRITICAL PLANNING FAILURE:** This more advanced step was likewise completely absent from the agent's plan.	✗ **FAIL**

Consequence of the Flawed Plan: By omitting two required data sources in the planning stage, the agent analyzed an incomplete and biased sample (9,073 records instead of the correct 11,838), thereby invalidating all subsequent analysis. This is a textbook Planning Error: once the initial strategy is faulty, execution quality cannot rescue the outcome.
Final score for this standard: 1 / 4.

Table 12: Focused case study of a critical *Execution Error*. This table examines the agent's implementation for Standard 2.1 (Channel Performance Metrics), illustrating how an otherwise sound plan partially failed due to an improper formula for a key metric.

Calculation Required by the Rubric	**Agent Implementation vs. Correct Method**	**Outcome**
Case: Standard 2.1 — Channel Performance Metrics		
Sub-standard 2.1.A.1: Compute `Sales Volume` by channel.	The agent correctly used 'GROUP BY' with 'SUM(sales_volume)'.	✓ **PASS**
Sub-standard 2.1.A.2: Compute `Total Revenue` by channel.	The agent correctly used 'GROUP BY' with 'SUM(total_revenue)'.	✓ **PASS**
Sub-standard 2.1.A.3/4: Compute `Average Unit Price` by channel.	**CRITICAL EXECUTION ERROR:** The agent treated unit price as a simple average rather than a revenue-weighted average. As a result, the reported average prices were incorrect and led to misleading conclusions about channel profitability.	✗ **FAIL**

Impact of the Execution Deviation: Although the agent's overall plan for channel analysis was sound, using the wrong formula for a single critical metric (Average Unit Price) produced misleading conclusions about channel profitability, directly undermining any price-based strategic recommendation. This constitutes a canonical *Execution Error*.
Final score for this standard: 5 / 6.

Table 13: Focused case study of a critical Interpretation Error. The table shows a stark contrast between the agent's successful execution of calculations and its failure to synthesize those results into a meaningful, context-aware conclusion.

Analytical Stage (from Rubric)	Agent's Performance & Justification	Outcome
Case Study: Analysis of Students with Suicidal Ideation		
Stage 1: Execution & Calculation (Standards 1.1 – 1.4)	The agent's plan was sound and its execution was flawless. It successfully filtered the correct data population and accurately calculated all required statistical metrics (e.g., average economic/academic pressure, lifestyle habit percentages).	✓ **PASS**
Stage 2: Interpretation & Synthesis (Standard 1.5: Create a "high-risk profile")	**CRITICAL INTERPRETATION FAILURE:** The agent failed to synthesize the previously calculated statistics into a coherent, higher-level insight. Instead of creating a "profile," the agent merely listed the numbers again. The judge noted the summary was **"not deep enough"** and **"merely restated the table's content."**	✗ **FAIL**
Stage 3: Contextual Understanding (Standard 2.2: Provide safety disclaimer)	**CRITICAL INTERPRETATION FAILURE:** The agent's final output completely omitted the mandatory "Limitations and Safety Disclaimer." This demonstrates a failure to understand the serious and sensitive context of the topic, which is a key part of providing a responsible and complete analytical deliverable.	✗ **FAIL**
Consequence of Flawed Interpretation: This case exemplifies a pure Interpretation Error. The agent acted as a perfect calculator, producing correct data (Stage 1). However, it failed at the final and most critical stage: transforming that data into a meaningful, insightful, and contextually appropriate conclusion (Stages 2 & 3).		

E ANNOTATION DETAILS

E.1 DATA COLLECTION

Data synthesis for de tables. Our DE tables originate from 73 enterprise-grade SaaS domains and their companion data-transformation projects, providing production-style schemas and realistic dependencies. Starting from a minimal business contract (target grain, primary/foreign keys, required metrics), we expand to end-to-end datasets and scale them while preserving business semantics and referential integrity. To keep the data mock both controllable and realistic, we highlight only the key steps: 1) *Schema fidelity*: retain PK/FK, uniqueness, not-null, and domain constraints; 2) *Distributions & dependencies*: fit marginal distributions and model conditional links (e.g., `country`⇒`currency`/`timezone`); 3) *Temporal coherence*: inject seasonality, trend, and holiday effects while maintaining fact–dimension integrity; 4) *Noise & edge cases*: introduce controlled missingness/outliers/type coercions and design stressors that expose pipeline fragility (e.g., duplicate "current" rows, currency conflicts, timezone mismatches). The synthesis pipeline is implemented in Python (pandas, numpy, faker) with custom generators to scale volume while honoring inter-column dependencies and business invariants.

E.2 CONSTRUCTION DETAILS OF DACOMP-DE

This subsection presents our experience constructing the DAComp-DE corpus. We outline an end-to-end process across three tracks—Architecture, Implementation, and Evolution—spanning the baseline derived from *73 enterprise-grade SaaS domains and their data-transformation projects* to pure-SQL normalization and validation, high-level requirement setting for blueprinting, contract-driven realization into working SQL, and change-oriented migration under realistic constraints. The summary reflects decisions and best practices agreed upon by domain experts to ensure rigor, reproducibility, and evaluability.

E.2.1 CONSTRUCTION DETAILS OF DACOMP-DE-ARCHITECTURE

Baseline curation and normalization. We first select open-source `dbt` projects that are license-compliant and empirically verified to be error-free, and normalize them into pure-SQL repositories by expanding materializations and macros while freezing model dependencies. Senior data engineers conduct a systematic audit of join semantics, analytical grains, window specifications, SCD handling, and testing assumptions, thereby establishing a high-quality baseline suitable for controlled evaluation.

High-level requirement formulation. Building on this baseline, we define task statements grounded in realistic enterprise scenarios: they provide only business context, overarching objectives, and expected outputs, without detailed metric definitions, precise calculation rules, or data constraint specifications. Such descriptions emphasize openness and cross-system characteristics, reveal gaps not covered by the existing repository, and intentionally avoid prescribing implementation paths or technical details. The model is expected to autonomously plan a blueprint—identifying key entities and dependencies, delineating layers and boundaries, and completing testing and freshness strategies—ultimately producing an executable architectural blueprint that evaluates its ability to plan end-to-end SQL projects and set constraints under incomplete information.

E.2.2 CONSTRUCTION DETAILS OF DACOMP-DE-IMPLEMENTATION

Contract formalization. DE-Impl is constructed by deriving a rigorous requirements specification from the vetted SQL baseline in the form of a standardized `data_contract.yaml` that follows enterprise conventions. The contract formalizes model inventory and lineage, table and column schemas with constraints, declared grains and time windows, metric definitions with coherent units and currency normalization, as well as data quality, freshness, and performance policies.

E.2.3 CONSTRUCTION DETAILS OF DACOMP-DE-EVOLUTION

Change specification. For DE-Evol, we start from a high-quality, production-style SQL repository and propose change requests driven by realistic enterprise pressures—such as revised metric definitions, altered analytical windows, schema drift, or governance hardening. Multiple experts specify

unambiguous business semantics, distinguish breaking from non-breaking changes, and design a safe migration plan that anticipates dependency revisions and testing upgrades.

E.3 Annotation details of DAComp-DA

In this section, we present the experience regarding the annotation of DAComp-DA data, which is summarized from our previous project discussion meetings and alignment meetings.

E.3.1 Core Design Principles

Strategic diversity The core of the Rubric is to evaluate problem-solving *strategies*, not *steps*. Each scoring Path must represent a methodologically distinct and self-contained solution. We avoid designing *complete* versus *abridged* versions of the same path. For example, *analyzing all provinces* and *analyzing a subset of provinces* should not be two separate Paths; the latter is merely an incomplete execution of the former.

Objective evaluation Scoring criteria must be quantifiable and reproducible to minimize scorer subjectivity. All items should be based on explicit evidence. Guideline: Any Accuracy item requiring numerical verification must have a pre-calculated Anchor Value. For open-ended paths without a single correct answer, a Pseudocode or a clear methodological verification process must be provided.

Dimensional separation of abilities Complex analytical skills are decomposed into independent scoring dimensions for a fairer and more granular assessment of model performance. Guideline: Strictly distinguish between *procedural execution* (were the steps completed?), *computational accuracy* (were the numbers correct?), and *insightful conclusion* (was the interpretation meaningful?), designing them as separate scoring items.

E.3.2 Structural components of the rubric

The Rubric employs a four-level hierarchical structure to deconstruct tasks, ensuring comprehensive and granular evaluation.

Requirement. Definition: The highest-level objective of the task, directly corresponding to a core analytical request from the user. Example: *Analyze the differences in employee attrition rates across departments and their causes.*

Standard. Definition: A key analytical step that must be completed or a core conclusion that must be reached to fulfill a Requirement. Example: *Standard 1: Calculate and verify the attrition rate differences between departments*; *Standard 2: Identify the key factors causing these differences.*

Path. Definition: A methodologically distinct and valid strategy for meeting a Standard. This is the core of the Rubric's design. Example: Under the standard of *verifying differences*, Path A could be *performing a statistical significance test (e.g., Chi-squared test)*, while Path B could be *making a descriptive statistical comparison (e.g., percentage difference)*.

Sub-standard / rubric item. Definition: The smallest scorable unit of the Rubric, nested under a specific Path and adhering strictly to the principle of dimensional separation. It comprises three main types:

- Completeness: Assesses whether all required steps for a given Path were executed. Focuses on *what was done*.
- Accuracy: Assesses whether the computational results or execution process are correct. Focuses on *if it was done correctly*. For deterministic paths, this is verified against an Anchor Value; for open-ended paths, it is verified against a methodological process or Pseudocode.
- Insightfulness: Assesses whether a reasonable and valuable conclusion or insight was derived from the correct results. Focuses on *if the results were understood*.

E.3.3 GOLDEN RULES FOR AUTHORS

These are the disciplinary requirements to ensure the quality and consistency of the Rubric. While these guidelines ensure consistency in creating rubrics for known strategies, the following section details our methodology for fairly evaluating novel or unanticipated solutions that may not align with pre-enumerated paths.

Calculate first, then author. Before finalizing the rubric, authors must personally run the complete analysis with code to calculate all Anchor Values required for the Accuracy assessment. This is the cornerstone of ensuring objective scoring.

Be specific and unambiguous. Every statement in the Rubric must be directive and unambiguous. Avoid subjective terms like *approximately*, *good*, or *relatively comprehensive* to minimize scorer discretion.

Avoid zero-point paths. If a method is not worthy of credit, it should not be designed as a distinct Path. A model's output that does not match any valid path will naturally receive no score for that standard.

F LLM USAGE DETAILS

In compliance with ICLR 2026 policies on large language model usage, we disclose that LLMs are mainly used for three purposes in this work:

- LLM Evaluation: The core of this work is the systematic benchmarking of various large language models to assess their performance and capabilities as data agents.
- LLM-based Judging: For the open-ended tasks in our benchmark, we employed an LLM as an automated judge to score agent responses based on a detailed, expert-designed rubric.
- Writing Assistance: We utilized an LLM to assist in polishing the manuscript by refining grammar, improving phrasing, and enhancing overall clarity.

All LLM outputs were subject to careful human oversight and validation. We take full responsibility for the accuracy and integrity of all content in this paper, including any sections enhanced with LLM assistance.

G DISCUSS

G.1 DISCUSSION OF HANDLING UNENUMERATED SOLUTION PATHS

Accuracy is the most critical dimension in our rubric. Since fully listing all valid analytical paths is often infeasible, we adopt a *three-tier, progressively relaxed* design for Accuracy: (i) *direct enumeration* with numeric anchors when the correct outcome can be exhaustively determined; (ii) *constrained computation* with pseudo-code anchors when procedures are well-defined but paths are not exhaustively enumerable; and (iii) *principle-based* assessment for highly open-ended cases.

Standardized assessment for common paths. We standardize scoring whenever we can verify correctness deterministically. *Tier-1 (numeric anchors):* for tasks whose outcomes can be *exhaustively enumerated*, we embed the reference value directly into the rubric (e.g., "How many users satisfy condition X?"), yielding absolute, reproducible checks. *Tier-2 (pseudo-code anchors):* for tasks with well-specified computation but multiple equivalent derivations (e.g., a conversion rate with alternative weighting schemes), we prescribe canonical steps in *pseudo-code* to constrain the procedure. This enables process-level verification (inputs, ordering, aggregation, null/edge handling) without enumerating every path, preserving both precision and reproducibility.

Principle-based assessment for novel paths. A minority of tasks are intrinsically open-ended, where enumeration or pseudo-code templating is impractical. Here we evaluate Accuracy via

methodological principles rather than a single anchor value. For example, a "key-driver identification" task may be solved by regression with coefficient interpretation (a pre-defined path), or by gradient boosting with SHAP attributions (an unenumerated path). We score such solutions on: *(1) Methodological appropriateness* (the method is suitable for the stated objective and data regime); *(2) Correctness of execution* (the pipeline is implemented soundly, with valid preprocessing, estimation, and validation); and *(3) Soundness of interpretation* (claims follow from the produced evidence, with clear caveats). This soft layer ensures valid but unconventional approaches are not penalized.

By construction, most DAComp items fall into Tiers 1–2, where numeric or pseudo-code anchors provide deterministic checks; Tier 3 is reserved for genuinely open-ended cases to maintain fairness without sacrificing rigor.

G.2 Discussion of Ambiguous of Requirements

Implementation and Evolution tasks in DAComp-DE are designed as deterministic evaluations. To balance *realism* with *unambiguous executability*, we adopt three principles:

1) Professionalism. Requirements are sourced from enterprise-style projects and vetted by senior data engineers for cross-layer impact, metric definitions, SCD handling, and temporal semantics. *Implementation* tasks emphasize canonical modeling pipelines from scratch; *Evolution* tasks mirror real "change requests" (e.g., metric revision, source replacement).

2) Unambiguity. *Implementation (node-first):* each SQL node has atomic contracts (schema, PK/-grain, time, nulls, joins, aggregation, SCD, idempotency). Multiple agents must converge under frozen contracts; discrepancies trigger tighter specifications. *Evolution (delta-first):* natural-language changes are mapped into minimal verifiable deltas (schema/logic/lineage), with explicit impact scope and before–after anchors; agent disagreement leads to refined deltas or explicit assumptions.

3) Realism. *Implementation:* converged nodes are composed into multi-node tasks, with contracts and assumptions documented (e.g., `data_contract.yaml`). *Evolution:* favors backward-compatible evolution (added columns/views, metric versioning); destructive changes require migration notes. All assumptions are logged for reproducibility.

G.3 Discussion on the Selection of Judger LLM

As shown in Tab. 4, both O4-Mini and gemini-2.5-flash achieve human-level agreement, while stronger proprietary models (e.g., gemini-2.5-pro, GPT-5) yield even higher consistency. For DAComp,, we standardize on gemini-2.5-flash, as it balances (1) cost efficiency for large-scale benchmarking, (2) stable and low-latency inference, (3) reproducibility across runsand (4) community accessibility. Choosing a widely available model ensures that our evaluation pipeline can be easily adopted, verified, and extended by others.

G.4 Discussion on End-to-End Evaluation

Current DAComp tasks span complementary stages of the data intelligence lifecycle: DE-Architecture (high-level specification and planning), DE-Implementation (multi-layer pipeline construction), DE-Evolution (safe modification under requirement changes), and DA (open-ended analysis over downstream data). Taken *together*, these stages delineate a strictly end-to-end process—from requirement articulation, through system realization and iterative evolution, to analytical insight and decision support—covering a full loop from planning and implementation to evolution and interpretation.

At present, we evaluate these stages modularly and in a decoupled fashion to enable controlled measurement at each step. Our next key objective is to integrate them into a single, end-to-end longitudinal evaluation: a single agent carries requirements through implementation and change propagation, and ultimately completes analysis and reporting. We contend this end-to-end setup offers substantial scientific and practical value: it stress-tests the *end-to-end consistency* of planning–execution–evolution–interpretation, better reflects real engineering workflows, and advances toward a comprehensive assessment of autonomous data agents' end-to-end capabilities.