Can AI Agents Answer Your Data Questions? A Benchmark for Data Agents

Ruiying Ma[1]*, Shreya Shankar[1]*, Ruiqi Chen[2], Adit Chintamaneni[1], Yiming Lin[1], Sepanta Zeighami[1], Rajoshi Ghosh[3], Abhinav Gupta[3], Anushrut Gupta[3], Tanmai Gopal[3], Aditya G. Parameswaran[1]

[1]UC Berkeley, [2]University of Washington, [3]Hasura PromptQL

{ruyingm, shreyashankar, adityagp}@berkeley.edu

Abstract

Users across enterprises increasingly rely on LLM agents to query their data through natural language. However, building reliable *data agents* remains difficult because real enterprise data is fragmented across many heterogeneous database systems, with duplicated and inconsistent data, and key information often buried in unstructured text—requiring agents to go beyond just writing SQL or data science scripts to answer questions. Existing benchmarks tackle only individual pieces of this end-to-end workflow—e.g., text-to-SQL over a single database—and are increasingly saturated and contaminated. We present a new benchmark for LLM agents, the Data Agent Benchmark (DAB), grounded in a study of enterprises building production data agents across six industries. DAB comprises 104 queries across 17 datasets and 4 database management systems. On DAB, the best state-of-the-art agent achieves only 57% pass@1. We analyze agent failure modes and distill takeaways for future data-agent development. Our benchmark and experiment code are published at github.com/ucbepic/DataAgentBench.

1 Introduction

Users across enterprises increasingly want *data agents*, or AI agents that answer natural-language questions over their data. Data agents are a major application of LLMs and a new workload pattern for data systems [31]: major database vendors have added agent capabilities to their platforms [12, 40], and organizations have invested heavily in building their own—for example, Uber's QueryGPT handles over 1.2 million interactive queries per month [43], and OpenAI built an internal data agent used by thousands of employees to query 70,000 datasets totaling 600 petabytes [44].

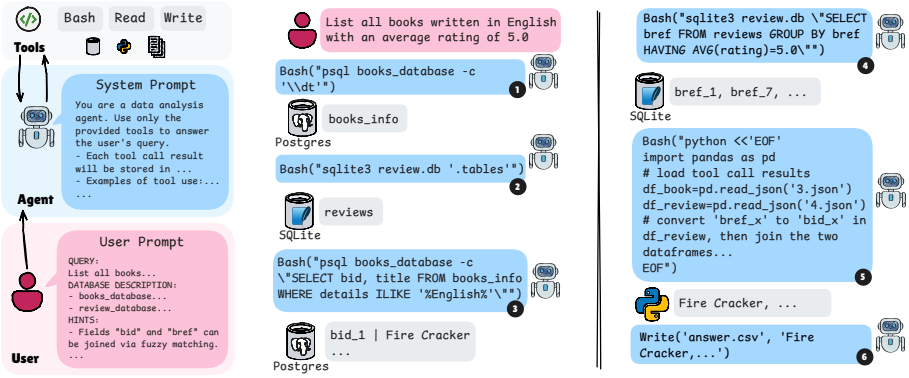
Yet building reliable data agents is hard. Consider an example query for a data agent: a sales analyst asks, *"Which leads from last quarter should we follow up on?"* Answering this requires finding lead records in a customer relationship management (CRM) tool, matching them against call transcripts stored in a separate document database, classifying each lead's intent from unstructured text, and applying domain knowledge about what makes a lead good to pursue. In fact, enterprise data is well-known to be fragmented and messy: surveys find that 72% of organizations store data in disparate silos [41] and 82% report that these silos disrupt critical workflows [19].

Unfortunately, no benchmark measures *end-to-end* data agent capabilities. For example, text-to-SQL benchmarks [8, 26, 28, 47] test whether LLMs can translate a natural-language question into a single correct query over a single relational database, but do not require multi-step reasoning or integration across different databases. Table question-answering (Table-QA) benchmarks [9, 10] provide tiny tables entirely in the prompt—unlike enterprise workloads, where tables live in databases and rarely fit in context. Moreover, the public benchmarks in this space are increasingly saturated by frontier models, and reported scores are further confounded by two issues: data contamination of public test sets in pretraining [38], and pervasive annotation errors that inflate accuracy—error rates reach 52.8% in BIRD Mini-Dev and 62.8% in Spider-Snow [21].

The Data Agent Benchmark (DAB). We present DAB, a new benchmark for evaluating AI agents on realistic, complex data-oriented tasks. **DAB *comprises 104 natural-language queries across 17 datasets and 4 database management systems (DBMSes).*** The size of DAB is comparable to other carefully-curated agent benchmarks (e.g., FrontierMath [13], TerminalBench [32]). Each DAB query requires an agent to interact with a

* Co-first authors.

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 42651–42677
October 24-29, 2026 ©2026 Association for Computational Linguistics



(a) DAB setup. (b) Example agent trace.

Figure 1: (a) In DAB, an agent answers a user query by interacting with tools provided by its harness (e.g., Claude Code, the OpenAI Codex SDK). (b) The agent extracts information embedded in a free-text column (`details`) via SQL, integrates data across PostgreSQL and SQLite, and reconciles ill-formatted join keys (`bref` vs. `bid`).

full environment using database and Python tools, potentially involving hundreds of reasoning and action steps. This differs substantially from large-scale benchmarks such as Text-to-SQL, where each item typically requires a single LLM call (e.g., to generate one SQL query). DAB has been publicly available[1] for five months and has already attracted 29 submissions.

Building DAB took us over a year and involved three core challenges. The first challenge was to solicit aspects of production workloads that make data-agent queries difficult. Through a formative study with six enterprises building production data agents (Section 3.1), we identified four properties that are unaddressed by existing benchmarks: *(i)* the data needed to answer a query is distributed across multiple heterogeneous database systems, *(ii)* identifiers for the same entity are formatted differently across databases, *(iii)* answers are buried in unstructured text fields and cannot be retrieved via SQL, and *(iv)* the query requires domain knowledge not inferable from the data alone (e.g., finance jargon).

The second challenge was to translate these properties into a reproducible benchmark. Since enterprise data is proprietary, we had to build DAB from open-source data—which is typically clean—to mirror the dirty, duplicated, and inconsistently formatted data we observed in production. We also had to ensure there is a deterministic, unambiguous ground-truth answer for each query.

The third challenge was calibrating difficulty: DAB should be hard enough to discriminate among agents, but not so hard that no agent succeeds. We spent $4,346 to evaluate three state-of-the-art frontier agents and harnesses—Claude-Sonnet-4.6 (Claude Code), GPT-5.5 (Codex), and Gemini-3.1-Pro (ReAct)—each under three experimental variants: no assistance, with database and schema descriptions added, and with both descriptions and hints describing how the data was corrupted. The best agent reaches only ***57% pass@1*** (5 trials per query), and most datasets show partial accuracy across agents, leaving headroom for improvement. We further analyze trajectories and provide a taxonomy of failure modes. 90% of wrong answers stem from incorrect planning or faulty implementation rather than wrong data selection; and agents rely almost exclusively on regular expressions for free-text extraction, with only one agent attempting alternatives like using NLP parsing libraries or spawning subagents to extract values in context.

In summary, this paper makes the following contributions:

1. We characterize real-world data-agent workloads via a study with six enterprises building data agents, and identify four properties that distinguish them from the queries existing benchmarks cover.
2. We construct and release DAB, a new benchmark of 104 queries across 17 datasets and 4 database systems.
3. We evaluate state-of-the-art agents on DAB and find that even the best agent achieves only 57% pass@1; our error analysis surfaces failure modes that point to concrete directions for improving data agents.

2 Related Work

Despite databases being among the most ubiquitous professional tools, no frontier-model evaluation in-

[1] https://ucbepic.github.io/DataAgentBench/

cludes a database-use benchmark. We position DAB against a few categories of benchmarks, summarized in Table 1.

Tool-use benchmarks. Tool-use benchmarks evaluate LLM agents on external tools such as function calling [37], code editing [20], browser interaction [51], command-line tasks [32], and multi-step web reasoning [33], but none involves querying a database. Closer to our setting, benchmarks for data-adjacent workflows such as retail customer service [46] and CRM operations [17] expose each data operation through a hand-crafted API endpoint—the agent never queries the underlying database directly.

Data-related benchmarks. Other benchmarks target data tasks directly, yet each captures only part of the data-agent workflow.

Table question-answering. Table QA benchmarks supply data in the prompt and ask the model to reason over in-context tables [9, 10, 35, 42, 52]; some add unstructured text, as in HybridQA [9], or domain knowledge, as in FinQA [10]. Because tables are given in context, the agent never queries a database, and real-world tables routinely exceed context-window limits.

Text-to-SQL. Text-to-SQL benchmarks instead have the model generate a query for a natural-language question over a single database—Spider [47], WikiSQL [50], and BIRD [28], with extensions to multi-turn dialogue [48] and cloud SQL dialects [26]. This setup is limited: it tests only SQL *generation* over a single clean database, whereas enterprise questions typically span multiple systems and require transformations that SQL cannot express. Even within this narrow setting, text-to-SQL performance on data-warehouse tasks over private data is abysmal [8]. These text-to-SQL benchmarks are also saturated and corrupted: top systems reach 91.2% execution accuracy on Spider [47], and the reported scores are confounded by data contamination—the public test sets appear in pretraining corpora [38]—and by annotation errors, with error rates of 52.8% in BIRD Mini-Dev and 62.8% in Spider 2.0-Snow [21].

Data science and engineering. Beyond SQL, other benchmarks task the agent with analyzing data, but over flat files (e.g., CSVs) rather than databases. DS-1000 [24], DA-Code [18], and KramaBench [23] require multi-step code over such files, while Spider 2-V [5] requires operating data-engineering tools through GUI actions (e.g., configuring an Airbyte sync). Neither setting requires querying a database directly.

Table 1: Coverage of DAB's four properties: (i) multi-database integration, (ii) ill-formatted join keys, (iii) unstructured text transformation, (iv) domain knowledge. ✔ yes, ○ partial, ✗ no.

Benchmark	(i)	(ii)	(iii)	(iv)
General tool-use [20, 37]	✗	✗	✗	○
Table QA [9, 10]	✗	✗	✔	○
Text-to-SQL [26, 28]	✗	✗	✗	○
Data science & eng. [18, 23]	✗	✗	✗	✗
Semantic query processing	✗	✗	✔	✗
DAB	✔	✔	✔	✔

Semantic query processing. Semantic query processing extends relational operators with natural-language variants powered by LLMs [22, 29, 30, 36, 39]. Benchmarks such as SemBench [25] and TAG-Bench [4] execute a pre-constructed pipeline of these operators over flat files to obtain a single answer, but do not capture multi-system integration. The semantic operations in DAB are a subset of those these benchmarks evaluate, embedded within an end-to-end querying workflow over real database systems.

Overall, each of these benchmarks covers only a slice of data work, and many prescribe the exact tools or operations the agent must use. DAB takes a different view: it specifies a realistic data setup—multiple databases with messiness seen in production workloads—and places no restrictions on how the agent solves the task.

3 Benchmark Construction

3.1 Formative Study

We conducted a study (approximately two months of interviews and analysis) with six enterprises building production data agents across six industries (technology, finance, food services, e-commerce, SaaS, and healthcare). The interview protocol and an anonymized summary of one enterprise's response are in Appendix A.1. We performed a thematic analysis [11] of the responses to find recurring patterns of challenges, and came up with the following four properties of the workloads that should be in DAB:

(i) **multi-database integration**: a single question may require querying across several databases with different query language dialects (e.g., SQL dialects and MongoDB's query language).

(ii) **ill-formatted join keys**: identifiers for the same entity may be formatted differently across databases—e.g., a book ID appearing as `bid_123`

Table 2: Overview of datasets and queries in DAB. DBMS: M=MongoDB, P=PostgreSQL, S=SQLite, D=DuckDB. #Tbl=tables, #Q=queries, #DK=queries requiring domain knowledge (property (iv)).

Dataset	#DBs	DBMS	#Tbl	#Q	#DK	Example Query
agnews	2	M,S	3	4	0	*What is the title of the sports article whose description has the greatest number of characters?*
bookreview	2	P,S	2	3	0	*Which English-language books in the 'Literature & Fiction' category have a perfect average rating of 5.0?*
civic_unstructured	2	M,S	2	10	5	*As of 2023-01-01, list all projects whose most recent report records both a type of 'capital' and a status of 'design' and that have accumulated total funding of more than $500,000.*
crmarenapro	6	D,P,S	27	13	10	*Which states have the quickest case closure time in the past 6 quarters?* (Assume today's date is 2022-10-26)
cve	4	D,M,P,S	6	10	10	*How many distinct CVEs published in 2023 are in CISA's KEV catalog AND have at least one vulnerable Apache product configuration?*
deps_dev_v1	2	D,S	3	2	2	*Among all NPM packages with license 'MIT' marked as release, which 5 have the highest GitHub fork count?*
github_repos	2	D,S	6	4	4	*Among repos not using Python, what proportion of README.md files include copyright information?*
googlelocal	2	P,S	2	4	0	*What are the top 5 businesses in Los Angeles, CA, ranked by highest average rating?*
imdb	2	P,S	6	10	3	*Among titles produced after 1990 with a US release date in the 1990s–2000s and a verified cast listing, what is the alphabetically first title?*
krama	3	M,S	4	10	8	*Among NIFC regions with a Geographic Area Coordination Center (GACC), which GACC accumulated the most human-caused wildfire acres from 2020–2023?*
music_brainz_20k	2	D,S	2	3	0	*Which store earned the most USD revenue from Brucqe Maginnis' song 'Street Hype' across all countries?*
pancancer_atlas	2	D,P	3	3	3	*Among alive BRCA patients, which top 3 histological types show the highest % of CDH1 gene mutations?*
patents	2	P,S	2	3	3	*Identify CPC areas with the highest EMA of patent filings (smoothing 0.2); return level-5 codes whose best year is 2022.*
stockindex	2	D,S	2	3	3	*Which stock index in Asia has the highest average intraday volatility since 2020?*
stockmarket	2	D,S	2754	5	5	*List all ETFs on NYSE Arca with adjusted close above $200 at any point in 2015; report the total count.*
usaspending	4	D,M,P,S	6	10	8	*How many distinct Department of Defense contract awards have an amount greater than $1,000,000?*
yelp	2	D,M	5	7	0	*During 2018, how many reviewed businesses offered either business parking or bike parking?*

in one table and `bref_123` in another—requiring the agent to detect and reconcile the mismatch before joining.

(iii) **unstructured text transformation**: answers may be embedded in text fields that the agent must parse into structured values before they can be filtered, grouped, or joined.

(iv) **domain knowledge**: answering the query correctly requires expertise not inferable from the data alone, such as knowing that stock volatility must be computed from adjusted closing prices to account for splits and dividends.

We exclude two common types of queries: those that depend on live external APIs, and open-ended queries (e.g., "why are my sales going down?"), since neither admits a single stable ground-truth answer.

3.2 Construction

Table 2 details the 17 datasets in DAB with a representative query each. Because the enterprises' data is proprietary, we build DAB from open-source datasets whose queries instantiate the four properties identified above. Every query in DAB involves (i) and at least one of (ii) or (iii); (iv) appears in proportion to its prevalence in the study. Constructing DAB took approximately eight months of author time, averaging roughly 50 hours per dataset to collect the source data, design and corrupt it, write the queries, and validate the ground-truth answers.

Datasets. Figure 2 illustrates the four-step dataset creation procedure; we walk through them, using the *bookreview* dataset as a running example.

(1) Collect open-source datasets. We collect open-source datasets that have multiple tables (so we can later distribute them across databases) and at least one unstructured text column (so we can later embed answer-revealing values into it). Four of the 17 datasets—*agnews* [49], *crmarenapro* [17], *imdb* [27], and *krama* [23]—come from existing benchmarks; we sample their queries (rewriting some to remove ambiguity). The remaining 14 are sourced from open data repositories. For example, *bookreview* consists of a `books_info` table (with an unstructured `description` field) and a separate `reviews` table.

(2) Induce ill-formatted join keys (property ii) and unstructured-text transformation (property iii). We remove columns that would trivially answer a query and re-embed their values elsewhere, so the agent must work to recover them. Four example strategies include:

- *Renaming a shared identifier across tables.* The same identifier appears under different column names in the two tables it joins. For example, in *bookreview*, the book identifier is `book_id` in the `books_info` table (PostgreSQL) but `purchase_id` in the `review` table (SQLite); to join the two, the agent must recognize that these columns refer to the same entity.
- *Applying many row-specific corruptions to a column.* For some datasets we apply one of several corruption methods row-by-row to the same col-

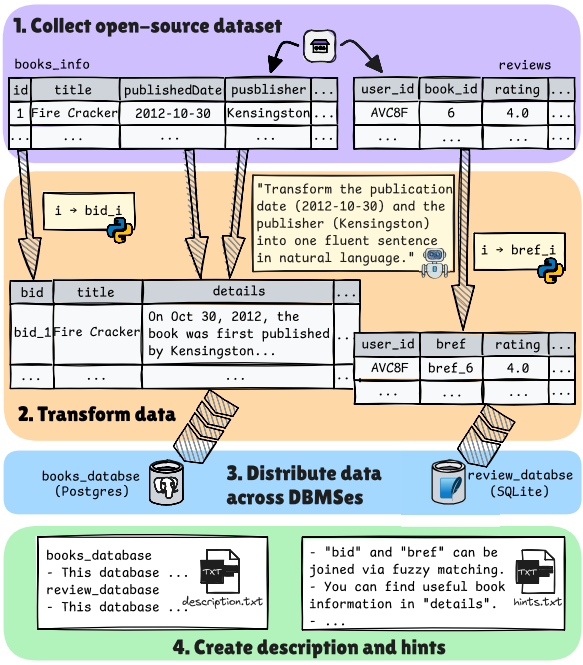


Figure 2: Dataset creation methodology, illustrated on the *bookreview* dataset. Step 1: collect an open-source dataset with two tables, `books_info` and `reviews`. Step 2: transform the data by removing `publishedDate` and `publisher` and re-embedding their values into a new `details` column via LLM-generated sentences, and by prefixing the join keys (`id` → `bid`, `book_id` → `bref`). Step 3: distribute the tables across PostgreSQL and SQLite. Step 4: create a dataset description (`descriptions.txt`) and a hints file (`hints.txt`).

umn, with so many variants per row that no small regex can clean the column: the agent has to inspect every row to figure out the cleaning logic. The methods include:

(a) prepending a different domain-relevant word to each value, generated by a coding agent (Claude Code) that produces thousands of plausible words per dataset. For *usaspending*, these words resemble agency codes, fiscal-year tags, or program identifiers (e.g., `FY2023_`, `DOD-`, `PROG/`).

(b) applying per-character OCR-like substitutions ($0 \leftrightarrow O$, $1 \leftrightarrow I$, $5 \leftrightarrow S$, etc.) with some probability ($\sim$0.1 per character).

We apply these methods independently in each table where the column appears. For example, in *usaspending*, the `award_id` column is corrupted with OCR substitutions, applied independently in the `contracts` and `recipients` tables; to join the two, the agent must essentially perform entity resolution on the `award_id` column.

• *Hiding answer-revealing values in a text column.* We prompt GPT-4o to rewrite a free-text column so it mentions the value in prose ("transform {free_text} to naturally include a reference to {value}; change as little as possible"), then drop the original column. For example, in *bookreview*, `publishedDate`, `publisher`, and `language` are embedded into `details` and removed from `books_info`.

• *Encoding a numeric column as varied free text.* We convert a numeric column into a free-text column where each row's value is written differently, requiring the agent to do data cleaning to recover the underlying number. For example, in *usaspending*, the contract award amount is stored as text where one row might read `1500000`, another `$1,500,000`, another `1.5M`, another `one million five hundred thousand dollars`, and another a sum-of-parts expression like `$1,050,000 + $450,000`.

(3) Distribute tables across DBMSes (property i: multi-database integration). We split each dataset's tables across at least two DBMSes, deciding placement manually to mirror how enterprises typically organize data: transaction-oriented tables (records, profiles, reviews) go in row-oriented or document stores (PostgreSQL, MongoDB, SQLite), analytics-oriented tables (large fact tables, time series) go in the columnar store DuckDB, and large free-text or document corpora (e.g., article bodies, review collections) go in MongoDB. The agent must therefore handle MongoDB's query language alongside SQL dialects. For example, *bookreview*'s book metadata table `books_info` goes in PostgreSQL and its `reviews` table in SQLite.

(4) Provide a natural-language description and hints file (the latter inducing property iv: domain knowledge). The description lists each database's logical name and schema; the hints file flags the transformations applied and any domain-specific definitions. Both are shown in the walkthrough in Section 3.3.

Queries and Answers. DAB contains 104 queries, each incorporating at least two of the four properties. A query comprises a natural-language question, a ground-truth answer (derived from the pre-transformation data), and a *validation script*: a Python function that takes the agent's free-form response, normalizes it (e.g., trimming whitespace, lower-casing), and returns True or False by checking whether the ground truth appears within the normalized string. Validation scripts are bespoke per queryaccording to the answer type as follow:

• Numeric answers: we check whether the response contains a value within a predefined tolerance range of the ground truth.

• Single-valued answers (e.g., a book title): the ground truth appears as a substring of the response.
• Alternative valid formats: some answers are semantically equivalent to the ground truth but formatted differently. For example, ETF names may appear with or without their share-class suffix (e.g., "Vanguard S&P 500 ETF" vs. "Vanguard S&P 500 ETF VOO"). Since these differ only slightly, we compute the Levenshtein distance between the response and the ground truth and accept it if below a threshold (on average, 20% of the answer's length).
• Set-valued answers (e.g., a list of book titles): every element must appear, each checked by the single-answer rule above.
• Dictionary-valued answers (e.g., book titles with their average ratings): every key must appear, and each value must appear within a surrounding window of its key, with keys and values checked by the single-answer rule above.

Our validation scripts check whether the response contains the complete correct answer, even if it also contains incorrect information. Detecting additional incorrect information would require manual review or an LLM judge, making evaluation not scalable or deterministic. Two authors jointly write each query, its ground truth, and its validation script. To verify DAB, we continue the following process until all reviewers reach consensus:

(i) Two authors independently review every dataset, solve every query, and test every validation script, using helpful tools (e.g., Claude for domain knowledge).

(ii) They compare their findings and reconcile disagreements. We classify disagreements into the following two types and resolve by root cause:
• Dataset or query disagreements arise almost entirely from ambiguous wording. We clarify the query and ask another author to solve each revision until the intended answer is unambiguous.
• Validation script disagreements arise when the script misses valid answer formats or edge cases. We identify the missing cases and refine the script to handle them (e.g., Levenshtein distance to tolerate minor string variations).

3.3 Statistics and Walk-Through

Statistics. Table 2 summarizes DAB's 17 datasets with a representative query each (full queries in Table 6). All 104 queries span multiple databases; the maximum is six databases across three systems (*crmarenapro*), and *stockmarket* navigates 2,754 tables. 73% of queries require reconciling ill-formatted join keys. 84% require extracting structured values from unstructured text. 62% require domain knowledge. Appendix A.4 maps each dataset and query to the four properties (Section 3.1) and the transformations applied (Section 3.2).

Walk-through. We illustrate DAB's structure using the *bookreview* dataset, which spans two databases (PostgreSQL and SQLite). Consider the query *"Which English-language books in the 'Literature & Fiction' category have a perfect average rating of 5.0?"* To answer it, the agent receives a natural-language description of each database and its schema:

Database Description (Excerpt)

You are working with two databases to solve this query. Here are the descriptions of these two databases:

1. books_database
System: PostgreSQL. Contains Amazon book information including descriptions, price, details, title, etc., up to 2023.
Tables:
- `books_info` (Book information): `title`, `subtitle`, `author`, `rating_number`, `features`, `description`, `price`, `store`, `categories`, `details`, `book_id`

2. review_database
System: SQLite. Contains Amazon book review information including ratings, review text, helpfulness votes, etc., up to 2023.
Tables:
- `review` (Review information): `rating`, `title`, `text`, `purchase_id`, `review_time`, `helpful_vote`, `verified_purchase`

(Descriptions for the table fields are omitted for brevity.)

It also receives a hints file describing the transformations applied during dataset creation, which flags data quality issues the agent may encounter:

Hints (Excerpt)

- `book_id` (in `books_info`) and `purchase_id` (in `review`) refer to the same book entities and can be matched via fuzzy join despite differences in formats.
- Some queries may require information from `details` or `categories` in `books_info`.

4 Experiments

We evaluate three frontier LLM agents on DAB (5 trials per query each); the best agent (GPT-5.5) reaches only 57% pass@1, no agent exceeds 70% pass@5, and one dataset (*patents*) is never solved. The rest of this section covers the setup (Section 4.1), main results (Section 4.2), ablations on dataset descriptions and hints (Section 4.3), and failure modes (Section 4.4). All agent trajectories are released[2].

[2] https://drive.google.com/file/d/1SjCkvwsc4m1S17l_rzu9PHAAei3jAL4i/view?usp=drive_link

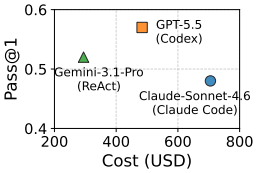


Figure 3: Total cost (USD) vs. pass@1 per agent.

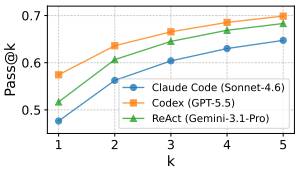


Figure 4: Pass@k as a function of k.

4.1 Setup

LLM Agents. We test three frontier LLMs, each paired with a state-of-the-art *harness*—an orchestration layer that wraps an LLM in a loop with some tools and default prompts, shown to substantially improve agent accuracy over fixed LLM API calls [1]. We enable all built-in tools for each harness. **Claude-Sonnet-4.6** (Anthropic) uses the Claude Code harness [2, 3], and **GPT-5.5** (OpenAI Azure Foundry) uses the Codex harness [34]. **Gemini-3.1-Pro** (Google)'s default harness, Gemini CLI, has no extensible SDK [14], so we implement the state-of-the-art ReAct harness [45] with four tools—`list_db`, `query_db`, `execute_python`, and `return_answer`; full tool signatures and prompt templates are in Appendix B. Individual tool calls time out after 600s; tool results larger than 10,000 characters are written to disk so agents can read them back. All parameters are set to provider defaults. We disable network access to prevent agents from loading external data (e.g., *agnews* ground-truth labels from Hugging Face) to obtain correct answers. For brevity, we hereafter name each agent by its model alone, omitting its harness.

Metrics. We measure accuracy via pass@k [7], with $n = 5$ trials per query per agent and c correct: pass@$k = 1 - \binom{n-c}{k} / \binom{n}{k}$. All averages are *stratified*—per query, then per dataset, then across the 17 datasets—so larger datasets don't dominate. We also report cost (USD) and *trajectory* statistics (iterations, tool calls). The evaluation totals 4,680 trials at $4,346.

4.2 Results

Our main experiments give each agent the full task context: the natural-language question, the database description, and the hints (Section 3.2). We also run ablations that remove the description and hints individually, presented in Section 4.3.

Accuracy. Figure 3 plots pass@1 for each agent. GPT-5.5 leads at 57%, followed by Gemini-3.1-Pro (52%) and Claude-Sonnet-4.6 (48%)—Claude is the least accurate despite being the most expensive.

Table 3: Per-agent pass@1 (Acc) and cost by dataset, in the main condition (description+hints). **Green**: highest pass@1; 0: zero pass@1.

	Claude-Sonnet-4.6		GPT-5.5		Gemini-3.1-Pro	
Dataset	**Acc**	**Cost**	**Acc**	**Cost**	**Acc**	**Cost**
agnews	**0.45**	$19.9	0.40	$16.0	0.25	$8.0
bookreview	**0.93**	$3.0	**0.93**	$3.1	**0.93**	$1.8
civic_unstructured	0.32	$123.8	0.20	$68.3	**0.34**	$30.2
crmarenapro	0.69	$25.9	0.68	$24.0	**0.75**	$17.5
cve	0.08	$269.3	**0.18**	$149.5	0.12	$95.2
deps_dev_v1	0	$3.2	**0.50**	$6.1	**0.50**	$2.9
github_repos	**0.45**	$5.9	0.25	$10.3	0.40	$5.6
googlelocal	0.50	$4.3	**1.00**	$4.0	0.80	$2.0
imdb	**0.76**	$28.2	0.62	$49.5	0.38	$12.5
krama	0.64	$105.8	**0.74**	$60.6	0.28	$47.8
music_brainz_20k	0.13	$2.3	0.67	$6.8	**0.73**	$2.4
pancancer_atlas	**0.60**	$5.1	**0.60**	$6.0	0.53	$1.8
patents	0	$7.2	0	$14.2	0	$4.5
stockindex	0.80	$2.2	**1.00**	$3.2	0.93	$1.7
stockmarket	0.80	$3.3	**0.84**	$3.6	0.80	$13.1
usaspending	0.34	$84.7	**0.50**	$46.9	0.40	$39.6
yelp	0.60	$11.3	**0.66**	$11.9	0.63	$7.6
Avg Acc; Total Cost	0.48	$705.4	**0.57**	$484.0	0.52	$294.3

Table 4: Trajectory statistics per agent. Each metric is first averaged across 5 trials for a given query, then across queries. The hardest queries for the agents all come from *cve*.

Agent	#LLM calls	#Tool calls	#DB queries	#Python execs
Stratified average across all queries				
Claude-Sonnet-4.6	30.5	30.3	10.3	11.2
GPT-5.5	16.3	12.7	9.2	2.3
Gemini-3.1-Pro	23.4	24.9	10.1	13.8
Hardest query (highest per-query average)				
Claude-Sonnet-4.6	160.0	159.0	38.5	102.5
GPT-5.5	44.4	40.0	14.0	21.4
Gemini-3.1-Pro	73.2	74.6	33.4	40.2

Table 3 reports per-dataset breakdowns; no agent solves any query in *patents*. Figure 4 plots pass@k as a function of k: agent rankings are stable, but the best agent still ***reaches only 70% pass@5***.

Cost-efficiency. Figure 3 plots total API cost against pass@1 (per-dataset breakdown in Table 3). Gemini-3.1-Pro offers the best cost–accuracy trade-off at $294 total (52% pass@1). GPT-5.5 costs 1.6× more for +5 pp accuracy, and Claude-Sonnet-4.6 costs 2.4× more yet is the least accurate. *cve* is the costliest dataset for every agent: it contains 95 MB of unstructured vulnerability records that agents pull into context to clean and join, and only GPT-5.5 and Gemini-3.1-Pro answer a few of its queries correctly.

Trajectory patterns. Table 4 reports iterations and tool calls per trajectory, broken down into database queries (`list_db`/`query_db` for ReAct; DB CLIs such as `duckdb`, `sqlite3`, `psql`, `mongosh`, or Python scripts importing only DB-related packages (`duckdb`, `sqlite3`, `psycopg2`, `pymongo`) for Claude Code and Codex) and Python executions (`execute_python` for ReAct; all other Python scripts for Claude Code and Codex). We highlight two takeaways.

Takeaway: Writing larger programs upfront is more cost-efficient than step-by-step exploration.

GPT-5.5 typically solves each query with one large Python program that inspects schemas, queries the database, and aggregates results in a single process. Claude-Sonnet-4.6 splits the same work into many smaller commands: a 0.9:1 DB-to-Python ratio versus 4:1 for GPT-5.5. For instance, on *cve*/query1, Claude-Sonnet-4.6 needs five Python executions to explore the database while GPT-5.5 does so in one. This pattern largely explains the cost gap: Claude-Sonnet-4.6 costs $705 at 48% pass@1, while GPT-5.5 costs $484 at 57%. Note that Gemini-3.1-Pro's DB-to-Python ratio is not directly comparable, since its ReAct harness exposes Python as a dedicated tool whereas Claude Code and Codex invoke Python via bash.

Takeaway: Parallel tool calling is underutilized, but could improve latency and cost.

Multi-database workloads in DAB would benefit from issuing tool calls in parallel, since each source can be queried independently. Exploratory calls (schema lookups, sample queries) are an especially natural fit for parallelization, and they make up a large fraction of trajectories: 51% for GPT-5.5, 39% for Claude-Sonnet-4.6, and 22% for Gemini-3.1-Pro (Table 11). A single iteration may issue multiple tool calls in parallel, which is why some agents in Table 4 have fewer iterations than tool calls. However, all three agents parallelize on fewer than 10% of turns: Claude-Sonnet-4.6 most often (8.2%), followed by GPT-5.5 (5.2%) and Gemini-3.1-Pro (5.1%). When they do parallelize, GPT-5.5 and Gemini-3.1-Pro occasionally issue large bursts—up to 16 and 100 concurrent calls in a single turn.

We also ran a shared-harness experiment to evaluate different models, pairing ReAct with GPT 5.2, Claude Opus 4.6, and Gemini 3.1 Pro. Due to budget limits, we used the 54 least expensive queries across 12 datasets, with five trials per model per query. Pass@1/pass@5 is 29%/44% for GPT 5.2, 55%/69% for Claude Opus 4.6, and 60%/75% for Gemini 3.1 Pro (details in Appendix C).

4.3 Ablations

The main experiments give each agent both dataset descriptions (databases, tables, schemas) and "hints" that describe our induced data quality issues. Here, we perform two ablations: removing hints only ("w/o hints") and removing both descriptions and hints ("w/o desc+hints"), with 5 trials per query in each.

Table 5: Average pass@1, total cost (USD), and per-trial tool calls under each ablation, with the difference vs. the full-info setting in parentheses. **Green**: highest pass@1 across agents per row.

Setting	Acc	Cost	#Tools
Claude-Sonnet-4.6			
w/o hints	0.50 (+0.02)	$670 (-$35)	29.0 (-1.3)
w/o desc+hints	0.43 (-0.05)	$686 (-$19)	37.9 (+7.6)
GPT-5.5			
w/o hints	**0.58** (+0.01)	$483 (-$1)	11.3 (-1.4)
w/o desc+hints	**0.55** (-0.02)	$453 (-$31)	15.8 (+3.1)
Gemini-3.1-Pro			
w/o hints	0.48 (-0.04)	$280 (-$14)	25.6 (+0.7)
w/o desc+hints	0.47 (-0.05)	$299 (+$5)	34.2 (+9.3)

Accuracy and cost. Table 5 reports the average pass@1, total cost, and per-trial tool calls under each ablation; the per-dataset breakdown is in Appendix D. Removing hints or descriptions sometimes decreases accuracy (as expected, since the agent has less information) and sometimes increases it: hints apply to the entire dataset, so when a particular query doesn't need a particular hint, the agent can be "misled". For example, on *crmarenapro*/query12, the agent sees a hint about join-key corruption—which actually applies to other queries in this dataset, not query12—and assumes its initial (correct) answer is wrong because it didn't clean for that corruption, so it reports a different, incorrect answer. Under "w/o hints", the agent has no such suspicion and answers correctly in 2/5 trials. Across the board, however, accuracy shifts are marginal: pass@1 moves by at most 5 pp in either direction. Costs also shift only modestly, between -6% and $+2\%$ across agents and conditions.

Tool use. Unsurprisingly, removing the description ("w/o desc+hints") drives all three agents to issue more tool calls on average—$+3$ to $+9$ per trial—as they probe schemas and surface data quality issues themselves. Removing hints alone has little effect (± 1 call). Interestingly, sometimes removing hints actually *decreases* exploration. For example, on *usaspending*, where a join column is corrupted in ways that require row-by-row reasoning, in the main experiment, agents consider the hint and issue many tool calls to do that reasoning (Claude-Sonnet-4.6: 53.5/trial, Gemini-3.1-Pro: 48.9/trial). But under "w/o hints", they never discover the corruptions, issue far fewer tool calls (39.4 and 22.7 respectively), and return the wrong answer.

4.4 Error Analysis

We also analyze failed trajectories to identify where the agents are breaking down.

Methodology. Two authors examined 30 failed trajectories from the *bookreview* dataset and defined five failure modes specific to data agents. To classify presence of failure modes at scale, we used GPT-5 as an LLM judgeover one incorrect trajectory per query per agent in the main experiment, yielding 196 annotated trajectories. The LLM judge doesn't perform open ended error analysis; it determines whether a trajectory exhibits each human defined failure mode, following prior work in agent evaluation [6, 32, 53]. We follow the procedure used by Terminal Bench. The full annotation prompt and per-agent breakdown are in Appendix E.2, and trajectories illustrating each FM appear in Appendix E.3. The failure modes are:

- **FM1** *Fails before planning*: the agent makes no attempt—either returning `None` as its tool call (**FM1 (no_tool_call)**) or refusing without invoking any tool (**FM1 (other)**).
- **FM2** *Incorrect plan*: the agent's plan cannot produce the correct answer even if executed perfectly—e.g., averaging book-level averages within a decade rather than averaging the underlying review ratings directly.
- **FM3** *Incorrect data selection*: the plan is right but the agent queries the wrong source—e.g., checking the `description` column for a book's language when the `details` column holds it.
- **FM4** *Incorrect implementation*: plan and data are right but execution is wrong—e.g., a regex intended to extract a publication year inadvertently matches an ISBN segment from `details`.
- **FM5** *Runtime error*: API failures (non-200 HTTP responses, tool-call array overflows, "service unavailable") or request timeouts (>600s).

Results. Figure 5 and Figure 6 in Appendix E.1 show the failure-mode breakdown across all 1,560 trajectories in the main experiment. We report two takeaways below.

Takeaway: Agents typically select the right data, but fail at planning the computation or implementing it correctly.

Among the trajectories that completed but returned incorrect answers (Figure 6), **FM4** (incorrect implementation) is the most common at 57%, followed by **FM2** (incorrect plan) at 31%, **FM3** (incorrect data selection) at 11%, and **FM1 (other)** at 1%. The low rate of **FM3** indicates that agents generally identify the correct tables and columns; the dominant challenge is deciding what to compute and computing it correctly. Together, **FM2** and **FM4** account for 90% of incorrect answers.

Takeaway: All agents rely on regex for text extraction and fail when it isn't expressive enough.

When agents need to parse text fields—e.g., pulling a date out of a sentence—they default to authoring and executing a regex; only Claude-Sonnet-4.6 ever calls an LLM-based extractor (and only on *agnews*). This breaks down on DAB's text-heavy datasets. For example, on *patents* (0% pass@1 for every agent), dates appear in many natural-language forms (e.g., "dated 5th March 2019", "March the 18th, 2019"), and no regex the agents write covers all of them. In another example, on *pancancer_atlas*, a regex for `MALE` matches inside `FEMALE` and mislabels gender. Perhaps exposing dedicated extraction tools (e.g., date parsers, NER, LLM-based extractors) as first-class tools, rather than forcing agents to author them ad hoc, could unlock many of DAB's hardest queries.

While our quantitative analysis is intended to compare the *relative* prevalence of the failure modes—rather than estimate the *exact* fraction of trajectories exhibiting each failure mode—to further adjudicate GPT-5's judge and validate the takeaways above, we ran two more experiments:

- Human validation: One author reviewed 30 trajectories, 10 per agent. 20 labels (67%) match the GPT-5 judge exactly; 3 (10%) partially match because the human assigns extra failure modes; and 7 (23%) disagree. Importantly, the human labels rank failure-mode prevalence the same as the judge.
- Agreement across LLM judges: We re-annotated all trajectories with GPT-5.4 to measure judge agreement. Cohen's κ is 0.80 for FM1 and 0.42–0.46 for FM2–FM4, indicating strong or moderate agreement. More importantly, both judges give the same ordering: FM2 (42% with GPT-5.4, 31% with GPT-5) and FM4 (44%, 57%) dominate, FM3 follows (13%, 11%), and FM1 is rare (1% for both).

5 Conclusion

We presented DAB, the first benchmark for evaluating data agents on realistic, multi-database queries. Grounded in a formative study of enterprise workloads, DAB comprises 104 queries across 17 datasets and 4 database systems. Even the best frontier agent achieves only 57% pass@1, and 90% of failures stem from incorrect plans or incorrect implementations rather than data selection. DAB represents a significant step forward for data agent evaluation, and we hope it helps the community build data agents that can be trusted in production.

6 Limitations

DAB reflects several tradeoffs we made during construction. First, DAB uses four open-source database systems (PostgreSQL, MongoDB, SQLite, and DuckDB) so that it can be run without commercial licenses; enterprise deployments commonly involve proprietary systems such as Oracle, Snowflake, and Databricks, which introduce additional challenges (e.g., vendor-specific SQL dialects, access-control policies) that DAB does not yet capture. Second, all queries are analytical and read-only—the benchmark does not cover transactional workloads in which agents must insert, update, or delete records. In fact, data systems typically use separate benchmarks for distinct workload classes rather than a single benchmark intended to cover them all. For example, analytical and transactional workloads are conventionally evaluated using TPC-H and TPC-C, respectively. Hex, a popular data agent startup, has multiple internal benchmarks for different workloads, such as warehouse analytics [16] and chart generation [15].

Third, every query has a single deterministic answer, which enables fully automatic evaluation but excludes open-ended analytical questions (Section 3.1) and queries requiring statistical modeling or machine-learning workflows; existing data science benchmarks [18, 23, 24] already target these capabilities. Fourth, DAB excludes queries that depend on external APIs (email clients, web search, third-party data providers); our formative study found these common in production, but live endpoints break reproducibility. Moreover, live external data introduce data quality issues that we cannot continuously identify and correct, making it difficult to evaluate agents fairly or ensure that the data meets the expectations established in our formative studies. Fifth, DAB contains 104 queries across 17 datasets—comparable in size to other carefully curated benchmarks (e.g., FrontierMath [13], TerminalBench [32])—but does not yet cover every domain or database system encountered in production. Despite manual verification and independent review by different authors (Section 3.2), some queries may be ambiguous or hints may contain inconsistencies. We plan to expand DAB over time and welcome community contributions and corrections. We hope the community develops benchmarks for additional data agent workloads, and we plan to extend DAB to cover more workloads over time.

7 Ethical Considerations

Our formative-study interviews and data-handling protocol were approved by our institution's IRB; participants gave informed consent, and we release no proprietary customer data, schemas, or queries. DAB is built from open-source datasets used consistent with their licenses (Section 3.2) and contains no PII. We release DAB's transformation scripts, validation scripts, and evaluation code under the MIT license; the transformed datasets are released for research use only, consistent with the licenses of their underlying sources.

We used LLMs at several points in this work—GPT-4o to embed structured attributes into free-form text during dataset construction (with all transformations author-verified to preserve deterministic ground truth), GPT-5 to annotate which failure modes appear in each incorrect trajectory (Figure 6), and Claude Haiku to classify tool calls as exploratory or analytical (Table 11).

The total experiment cost is $4,346 (Section 4.2). This is a substantial figure, especially for university research labs, and it is a real barrier to reproducibility: re-running the full evaluation, or evaluating a new agent on DAB, takes thousands of dollars. To partially mitigate this, we release all agent trajectories and code so that downstream analyses can be performed without re-running the experiments.

We caution that accuracy on DAB does not imply deployment readiness, particularly in regulated domains.

Acknowledgments

We acknowledge support from grants DGE-2243822, IIS-2129008, IIS-1940759, and IIS-1940757 awarded by the National Science Foundation, funds from the State of California, funds from the Alfred P. Sloan Foundation, as well as EPIC lab sponsors: Adobe, Google, G-Research, Microsoft, PromptQL, Sigma Computing, Snowflake, Bridgewater, and Jane Street. Compute credits were provided by Azure, Modal, NSF (via NAIRR), OpenAI.

References

[1] Anthropic. Raising the bar on SWE-bench Verified with Claude 3.5 Sonnet. https://www.anthropic.com/engineering/swe-bench-sonnet, 2024. Accessed: 2026-05-24.

[2] Anthropic. Claude code. https://docs.anthropic.com/en/docs/agents-and-tools/claude-code/overview, 2025. Accessed: 2026-02-28.

[3] Anthropic. Claude Agent SDK. https://docs.claude.com/en/api/agent-sdk/overview, 2025. Accessed: 2026-05-24.

[4] Asim Biswal, Liana Patel, Siddarth Jha, Amog Kamsetty, Shu Liu, Joseph E Gonzalez, Carlos Guestrin, and Matei Zaharia. Text2sql is not enough: Unifying ai and databases with tag. *arXiv preprint arXiv:2408.14717*, 2024.

[5] Ruisheng Cao, Fangyu Lei, Haoyuan Wu, Jixuan Chen, Yeqiao Fu, Hongcheng Gao, Xinzhuang Xiong, Hanchong Zhang, Wenjing Hu, Yuchen Mao, et al. Spider2-v: How far are multimodal agents from automating data science and engineering workflows? *Advances in Neural Information Processing Systems*, 37:107703–107744, 2024.

[6] Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

[7] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. 2021.

[8] Peter Baile Chen, Fabian Wenz, Yi Zhang, Devin Yang, Justin Choi, Nesime Tatbul, Mike Cafarella, Çağatay Demiralp, and Michael Stonebraker. BEAVER: An enterprise benchmark for text-to-SQL. In *The 4th Table Representation Learning Workshop at ACL 2025*, 2025. URL https://openreview.net/forum?id=OgP25r2pBZ.

[9] Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. Hybridqa: A dataset of multi-hop question answering over tabular and textual data. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1026–1036, 2020.

[10] Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan R Routledge, et al. Finqa: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3697–3711, 2021.

[11] Victoria Clarke and Virginia Braun. Thematic analysis. *The journal of positive psychology*, 12(3):297–298, 2017.

[12] Databricks, Inc. Introducing Databricks assistant data science agent. https://www.databricks.com/blog/introducing-databricks-assistant-data-science-agent, 2025. Accessed: 2026-02-28.

[13] Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, Caroline Falkman Olsson, Jean-Stanislas Denain, Anson Ho, Emily de Oliveira Santos, et al. Frontiermath: A benchmark for evaluating advanced mathematical reasoning in ai. *arXiv preprint arXiv:2411.04872*, 2024.

[14] Google Gemini Team. feat: Support Gemini CLI as a library (#2023). https://github.com/google-gemini/gemini-cli/issues/2023, 2025. Accessed: 2026-05-24.

[15] Hex. Making ai charts go brrrr. https://hex.tech/blog/making-ai-charts-go-brrrr/, 2026. Accessed: 2026-08-26.

[16] Hex. How we built a lab to evaluate data agents. https://hex.tech/blog/evaluate-data-agents/, 2026. Accessed: 2026-08-26.

[17] Kung-Hsiang Huang, Akshara Prabhakar, Sidharth Dhawan, Yixin Mao, Huan Wang, Silvio Savarese, Caiming Xiong, Philippe Laban, and Chien-Sheng Wu. Crmarena: Understanding the capacity of llm agents to perform professional crm tasks in realistic environments. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3830–3850, 2025.

[18] 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*, pages 13487–13521, 2024.

[19] IBM. What are data silos? https://www.ibm.com/think/topics/data-silos, 2024. Accessed: 2026-02-03.

[20] 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 Twelvth International Conference on Learning Representations*, 2024.

[21] Tengjun Jin, Yoojin Choi, Yuxuan Zhu, and Daniel Kang. Pervasive annotation errors break text-to-sql benchmarks and leaderboards. *arXiv preprint arXiv:2601.08778*, 2026.

[22] Saehan Jo and Immanuel Trummer. Thalamusdb: Approximate query processing on multi-modal data. *Proceedings of the ACM on Management of Data*, 2 (3):1–26, 2024.

[23] Eugenie Lai, Gerardo Vitagliano, Ziyu Zhang, Om Chabra, Sivaprasad Sudhir, 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. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=fZfUdeCC5X.

[24] 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*, pages 18319–18345. PMLR, 2023.

[25] Jiale Lao, Andreas Zimmerer, Olga Ovcharenko, Tianji Cong, Matthew Russo, Gerardo Vitagliano, Michael Cochez, Fatma Özcan, Gautam Gupta, Thibaud Hottelier, et al. Sembench: A benchmark for semantic query processing engines. *arXiv preprint arXiv:2511.01716*, 2025.

[26] 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. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[27] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? *Proceedings of the VLDB Endowment*, 9(3):204–215, 2015.

[28] 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*, 36:42330–42357, 2023.

[29] Paweł Liskowski, Benjamin Han, Paritosh Aggarwal, Bowei Chen, Boxin Jiang, Nitish Jindal, Zihan Li, Aaron Lin, Kyle Schmaus, Jay Tayade, et al. Cortex aisql: A production sql engine for unstructured data. *arXiv preprint arXiv:2511.07663*, 2025.

[30] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, Rana Shahout, et al. Palimpzest: Optimizing ai-powered analytics with declarative query processing. In *Proceedings of the Conference on Innovative Database Research (CIDR)*, page 2, 2025.

[31] Shu Liu, Soujanya Ponnapalli, Shreya Shankar, Sepanta Zeighami, Alan Zhu, Shubham Agarwal, Ruiqi Chen, Samion Suwito, Shuo Yuan, Ion Stoica, Matei Zaharia, Alvin Cheung, Natacha Crooks, Joseph E. Gonzalez, and Aditya G. Parameswaran. Supporting our AI overlords: Redesigning data systems to be agent-first. In *16th Annual Conference on Innovative Data Systems Research (CIDR)*, 2026.

[32] Mike A Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026.

[33] Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023.

[34] OpenAI. Codex cli. https://github.com/openai/codex, 2025. Accessed: 2026-05-24.

[35] Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1470–1480, 2015.

[36] Liana Patel, Siddharth Jha, Melissa Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. Semantic operators and their optimization: Enabling llm-based data processing with accuracy guarantees in lotus. *Proceedings of the VLDB Endowment*, 18(11):4171–4184, 2025.

[37] Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

[38] Federico Ranaldi, Elena Sofia Ruzzetti, Dario Onorati, Leonardo Ranaldi, Cristina Giannone, Andrea Favalli, Raniero Romagnoli, and Fabio Massimo Zanzotto. Investigating the impact of data contamination of large language models in text-to-sql translation. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

[39] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G Parameswaran, and Eugene Wu. Docetl: Agentic query rewriting and evaluation for complex

document processing. *Proceedings of the VLDB Endowment*, 18(9):3035–3048, 2025.

[40] Snowflake, Inc. Cortex agents. https://docs.snowflake.com/en/user-guide/snowflake-cortex/cortex-agents, 2025. Accessed: 2026-02-28.

[41] Stitch. The causes and costs of data silos. https://www.stitchdata.com/resources/data-silos/, 2024. Accessed: 2026-02-03.

[42] Alon Talmor, Ori Yoran, Amnon Catav, Dan Lahav, Yizhong Wang, Akari Asai, Gabriel Ilharco, Hannaneh Hajishirzi, and Jonathan Berant. Multimodal{qa}: complex question answering over text, tables and images. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=ee6W5UgQLa.

[43] Uber Engineering. Querygpt – natural language to sql using generative ai. https://www.uber.com/blog/query-gpt/, September 2024. Uber Engineering blog post; accessed Feb 2026.

[44] Brandon Xu et al. Inside OpenAI's in-house data agent. https://openai.com/index/inside-our-in-house-data-agent/, January 2026.

[45] 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.

[46] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. τ-bench: A benchmark for Tool-Agent-User interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[47] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[48] Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, et al. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 1962–1979, 2019.

[49] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

[50] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

[51] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2024.

[52] Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. Tat-qa: A question answering benchmark on a hybrid of tabular and textual content in finance. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3277–3287, 2021.

[53] Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, et al. Agent-as-a-judge: Evaluate agents with agents. *arXiv preprint arXiv:2410.10934*, 2024.

A Benchmark Construction Details

A.1 Interview Protocol and Sample Response

We solicited each enterprise's data-agent workload through a structured questionnaire. For each enterprise, we asked the analysts and engineers building the data agent to provide:

1. the natural-language queries their end users pose (or wish to pose) to a data agent;
2. the schemas of their data sources, including the database system each source uses (e.g., PostgreSQL, MongoDB, Salesforce, vector store);
3. where each piece of data is stored, including overlapping or redundant data across systems;
4. the join keys between tables, both within a single database and across databases; and
5. for each example query, a plan describing how an analyst would manually perform the joins, transformations, and any text extractions needed to answer it.

Below we reproduce an anonymized sample response from one enterprise, an enterprise project-management SaaS platform (comparable in scope to products such as Asana or Monday).

Data sources.

1. **Platform database** (PostgreSQL): `customers`, `users`, `projects`, `tasks`, `feature_usage`.
2. **Support database** (PostgreSQL): `agents`, `tickets`, `ticket_responses`.
3. **Salesforce**: `accounts`, `opportunities`, `contacts`.
4. **Billing database** (MongoDB): `invoices`, `usage_metrics`.
5. **Documentation** (vector database): `documents` with text and embeddings.

The response also provided example rows and join keys for each source (e.g., `customers.customer_id` ↔ `users.customer_id`; Salesforce `accounts.Name` fuzzily matches `customers.company_name`).

Example queries. The response provided dozens of queries organized into five categories. We list a representative example from each, with the analyst's stated plan.

(1) Joining across data sources. *Exact (1:1) join.* "Which enterprise-plan customers have raised a support ticket in the last 30 days?" *Plan:* filter `customers` by `plan_type=enterprise`, join with `users` on `customer_id`, query `tickets` by these `user_id` s with `created_date` in the last 30 days.

Programmatic transform join. "Which Salesforce accounts correspond to our platform customers?" *Plan:* normalize company names in both `customers` and Salesforce `accounts` (strip "Inc", "Corp" suffixes; lowercase); join on normalized names.

Fuzzy/semantic join. "Which customers mentioned in Salesforce opportunities are the same as our existing platform customers?" *Plan:* fuzzy string matching to reconcile variations such as "International Business Machines" vs. "IBM".

(2) Bulk semantic operations. *Classification.* "Does our average ticket resolution time differ for production vs. non-production issues?" *Plan:* classify each ticket from `tickets.description` using an LLM; aggregate `resolution_time` per class.

Extraction. "Which software versions have reported the most issues?" *Plan:* extract version numbers from ticket descriptions (e.g., "v2.1.3") with regex/NER; group by version.

Clustering. "What are the top issues that keep recurring?" *Plan:* semantically cluster ticket descriptions and rank clusters by frequency.

Generation/summarization. "Generate a monthly support performance report with key insights."

(3) Search. "If someone is running into XYZ error, what should we do?" *Plan:* vector-search the documentation database and the resolved-ticket archive; combine results.

(4) Domain knowledge. "Which deals are 'at risk'?" *Plan:* apply this company's definition of "at risk"—opportunities with `Probability < 50%` closing within 30 days, or accounts with billing issues or declining usage metrics.

(5) Open-ended. "What should I do to improve my support process?" (Excluded from DAB because no single correct answer.)

A.2 All Queries in DAB

Table 6 lists all queries in DAB. The raw data are available at https://github.com/ucbepic/DataAgentBench.

Table 6: All queries in DAB

Dataset	Query
agnews	*(1) What is the title of the sports article whose description has the greatest number of characters?*
	(2) What fraction of all articles authored by Amy Jones belong to the Science/Technology category?
	(3) What is the average number of business articles published per year in Europe from 2010 to 2020, inclusive?
	(4) In 2015, which region published the largest number of articles in the World category?
bookreview	*(1) Which decade of publication (e.g., 1980s) has the highest average rating among decades with at least 10 distinct books that have been rated? Return the decade with the highest average rating.*
	(2) Which English-language books in the 'Literature & Fiction' category have a perfect average rating of 5.0? Return all matching books.
	(3) Which books categorized as 'Children's Books' have received an average rating of at least 4.5 based on reviews from 2020 onwards?
civic_unstructured	*(1) As of 2023-01-01, list all projects whose most recent report (dated on or before 2023-01-01) records both a type of 'capital' and a status of 'design' and that have accumulated total funding of more than $500,000. Accumulated total funding means the sum of all funding amounts granted to that project on or before 2023-01-01.*
	(2) As of 2023-01-01, among all projects whose most recent report (dated on or before 2023-01-01) records a status of 'completed', which project has the highest accumulated total funding, and what is that funding amount in USD?
	(3) As of 2022-08-01, list all funding sources where at least 25% of the distinct projects they have funded on or before that date have a status of 'completed' in their most recent report dated on or before 2022-08-01.
	(4) What is the ratio of total funding amount granted to capital projects versus disaster projects between 2021-06-01 and 2022-06-01 (inclusive)? For each funding record, use the project's type as of the date that funding was granted. Express as a single decimal number rounded to 2 decimal places (capital_total / disaster_total).
	(5) Between 2021-06-01 and 2023-01-01, for all projects that underwent at least one status change within this period, what is the average total funding received per project? Report the average in USD, rounded to 2 decimal places.
	(6) Across all years in the funding records, which year did capital projects receive the highest total funding amount? Report the year and the total funding amount in USD.
	(7) As of 2022-11-01, list all projects whose most recent report records both a type of 'disaster' and a status of 'not started' and that have accumulated total funding of more than $1,000,000.
	(8) As of 2022-06-01, among all projects whose most recent report records a type of 'capital' and a status of 'construction', which project has the lowest accumulated total funding, and what is that funding amount in USD?
	(9) As of 2022-11-01, list all funding sources where more than 55% of the distinct projects they have funded have a type of 'disaster' in their most recent report dated on or before 2022-11-01.
	(10) What is the ratio of total funding granted to projects with a status of 'completed' versus projects with a status of 'not started' between 2021-09-01 and 2022-09-01 (inclusive)? Express as a single decimal number rounded to 2 decimal places (completed_total / not_started_total).
crmarenapro	*(1) Can this lead be qualified based on the latest discussions? If the answer is no, which factors—'Budget', 'Authority', 'Need', or 'Timeline'—are responsible? Return only one or several of the four BANT factors that the lead qualification fails to meet (i.e. 'Budget', 'Authority', 'Need', 'Timeline').* *## Lead qualification guide.* *Look for the voice call transcripts with the lead and relevant knowledge articles to justify the lead qualification.* *- Lead Id to be considered is: 00QWt0000089AekMAE*
	(2) Does the cost and setup of this quote comply with our company policy? If it doesn't, which knowledge article is it in conflict with? Return only the Id of the knowledge article that the quote violates. If no violation is found, return None. *## Quote approval guide.* *Look for relevant knowledge articles to justify the quote approval.* *- Quote Id to be considered is: 0Q0Wt000001WSDVKA4*
	(3) Is the stage name accurately representing the tasks for this opportunity? If it is not, what should the appropriate stage name be? Return only the correct stage label among ('Qualification', 'Discovery', 'Quote', 'Negotiation', 'Closed/). *- Opportunity Id to be considered is: 006Wt000007BGGjIAO*
	(4) Is there a particular month in the past 10 months where the number of SecureAnalytics Pro cases significantly exceeds those of other months? The associated product Id is 01tWt000006hVJdIAM. Return only the month name. *- Today's date: 2021-04-10*
	(5) What has been the most frequent problem AI Cirku-Tech encountered over the past five months? The associated product Id is 01tWt000006hV8LIAU. Return only the issue Id of the most reported issue for this product. *- Today's date: 2023-01-16*

Table 6 (continued)

Dataset	Query
	(6) Is the product setup in this quotation, including elements like quantity and price, against company regulations? Return only the Id of the knowledge article that the invalid config violates. *## Invalid config guide.* *Look for the relevant knowledge articles to justify the invalid config.* *- Quote Id to be considered is: 0Q0Wt000001WRAzKAO*
	(7) Did the agent breach the policy, and if so, which knowledge article was breached? Return only the Id of the knowledge article or None if no violation is found. *- Case Id to be considered is: 500Wt00000DDyznIAD*
	(8) Identify the agent with the fewest transfer counts in the last 4 quarters among those who handled more than 0 cases. Return only the Id of the agent. *## Transfer Count Policy* *- Definition: The number of instances a case was reassigned or transferred from one agent to another. Each transfer from agent A to agent B adds to the transfer count for agent A.* *- In the queries that specify 'agents managed/queries x cases'—this filter applies to both the first agent that the case was first assigned to and the agent that the case was transferred to. This means that if an agent has 2 cases that was initially assigned to itself by admin and 1 case transferred from another agent, a filter like 'handled/managed at least 3 cases" would not filter this agent out.* *- For cases that have NOT been transferred to an other agent, there will be only ONE 'Owner Assignment', and for those that have been transferred, there will be MORE THAN ONE 'Owner Assignment'.* *## Today's date: 2023-04-10*
	(9) Which states have the quickest case closure time in the past 6 quarters? Return only the two-letter abbreviation of the most matching state (eg. CA). *- Today's date: 2022-10-26*
	(10) In the past four months, which agent had the lowest average handle time for those processing more than one case? Return only the Id of the agent. *## Handle Time Policy* *- Definition: The duration taken to close a case. Specifically, it is the time from when a case is opened to when it is closed.* *- In the queries that specify 'agents managed/queries x cases'—this filter applies to both the first agent that the case was first assigned to and the agent that the case was transferred to. This means that if an agent has 2 cases that was initially assigned to itself by admin and 1 case transferred from another agent, a filter like 'handled/managed at least 3 cases' would not filter this agent out.* *- When computing handle time, we do not compute handle time for cases that have been transferred to other agents.* *- For cases that have NOT been transferred to an other agent, there will be only ONE 'Owner Assignment', and for those that have been transferred, there will be MORE THAN ONE 'Owner Assignment'.* *## Today's date: 2023-09-02*
	(11) Can you show me the AI processing unit I purchased last month? Return only the Id of the product from the contact's relevant past transaction. *- Contact Id interacting: 003Wt00000Jqy8SIAR* *- Today's date: 2021-07-15*
	(12) Who had the quickest average turnaround from opening to closing opportunities among agents in April 2023? Return only the Id of the agent. *## Sales Cycle Policy* *- Definition: The sales cycle is measured as the number of days between an opportunity's creation date and the company signed date on the corresponding contract.* *## Today's date: 2024-09-12*
	(13) Identify the agent who achieved the top sales figures for orders made in the past five months. Return only the Id of the agent. *## Sales Amount Policy* *- Definition: The sales amount for an order is calculated as the product of the quantity and the unit price from the Order object (Quantity * UnitPrice). An opportunity is eligible if its associated contract has a company signed date that falls within the time interval of interest.* *- Today's date: 2022-11-25*
cve	*(1) How many distinct CVEs published in 2023 are listed in CISA's Known Exploited Vulnerabilities (KEV) catalog AND have at least one vulnerable Apache product configuration?*
	(2) Among CVEs in CISA's KEV catalog flagged as having known ransomware-campaign use, identify the canonical vendor (using kev_vendor_aliases to resolve KEV vendor_project surface forms) whose CVEs include the most CRITICAL-severity entries (consider only vendors with at least 3 KEV-ransomware-flagged CVEs).
	(3) How many entries in CISA's KEV catalog reference a CVE identifier for which there is NO matching record in the NVD vulnerability database (vulns_db)?
	(4) For each canonical vendor that has at least 10 distinct CVEs both listed in CISA's KEV catalog AND with at least one vulnerable CPE configuration, compute the share of those CVEs whose CVSS v3 base score is 9.0 or higher. Return the vendor with the highest share, breaking ties alphabetically.

Table 6 (continued)

Dataset	Query
	(5) Among CVEs in CISA's KEV catalog that affect Apache Tomcat (canonical vendor 'apache', product 'tomcat'), how many distinct affected version strings are recorded, excluding wildcard ('') and null values? Treat versions differing only in encoding as the same version (e.g., '9.0.71', '9,0,71', and 'v9_0_71').*
	(6) How many distinct CVEs have NO English-language description, DO have at least one non-English description, AND are listed in CISA's KEV catalog?
	(7) Among CPE match entries flagged as vulnerable AND whose CVE is listed in CISA's KEV catalog, which canonical (vendor, product) pair has the highest count of rows? Return in the form 'vendor:product' (both lowercase).
	(8) How many distinct CVEs are listed in CISA's KEV catalog AND have at least one vulnerable Microsoft product configuration AND have a CVSS v3 base score of 7.0 or higher?
	(9) How many distinct CVEs appear in more than one row of the cves table in vulns_db with conflicting cvss3_attack_vector values AND are listed in CISA's KEV catalog?
	(10) Among the products listed in CISA's KEV catalog whose canonical vendor is Microsoft, how many distinct products have at least one associated CVE with a CVSS v3 base score of 9.0 or higher? Split rows whose product field contains comma-separated lists into individual products, and count case-insensitively.
deps_dev_v1	*(1) Considering only the latest release versions for each distinct NPM package, which packages are the top 5 most popular based on the Github star number, as well as their versions?*
	(2) Among all NPM packages with project license 'MIT' and marked as release, which 5 projects have the highest GitHub fork count?
github_repos	*(1) Among repositories that do not use Python, what proportion of their README.md files include copyright information?*
	(2) Identify the repository in Swift language that contains the most frequently copied non-binary Swift file in the dataset, ensuring that each file is uniquely determined by its ID.
	(3) How many commit messages are found in repositories that use the Shell programming language and are licensed under Apache-2.0, where each message exists, is shorter than 1,000 characters, and does not begin with 'merge', 'update', or 'test'?
	(4) List the repository names for the top five GitHub repositories whose main language is not Python, ordered by the highest number of commits.
googlelocal	*(1) What are the top 5 businesses located in Los Angeles, California, ranked by highest average rating in descending order?*
	(2) Which massage therapy businesses have an average rating of at least 4.0, and what are their respective average ratings?
	(3) What are the top 5 businesses that remain open after 6:00 PM on at least one weekday, ranked by highest average rating? Include their names, operating hours, and average ratings.
	(4) Which 3 businesses received the highest number of reviews with ratings of 4.5 or higher during 2019? Include their names and the count of high-rating reviews.
imdb	*(1) Among movies, TV movies, video movies, and video games produced after 1990 that are tagged with at least one keyword, are associated with a US company, have a US internet release date recorded in the 1990s or 2000s, and have a complete and verified cast listing—what is the alphabetically first title, and what type of title is it?*
	(2) Among writers who have a registered pseudonym and are credited on movies associated with a US company, what is the alphabetically first pseudonym and the alphabetically first movie title?
	(3) Among female actresses credited with a voice role—including general voice, uncredited voice, Japanese dubbed version, or English dubbed version—in movies produced after 2000 that are associated with a US company and have a release date on record, where the actress has a registered alternate name and played a named character, what is the alphabetically first actress name and the alphabetically first movie title?
	(4) Among cast and crew members whose name starts with 'Z' and who are credited in movies tagged with the keyword 'character-name-in-title' that are associated with at least one company, what is the alphabetically first name?
	(5) Among titles of kind 'movie' that have a release date and a rating on record and are associated with a US production company, what are the alphabetically first company name, the alphabetically first rating value, and the alphabetically first title?
	(6) Among female actresses whose name contains 'An', who have a registered alternate name, played a named character, and have trivia on record, and who are credited with a voice role in a computer-animated film produced between 2000 and 2010 with a Japan or USA release, associated with a US company—what are the alphabetically first character name, actress name, and title?
	(7) Among cast and crew members who have a registered pseudonym and are credited in movies tagged with the keyword 'character-name-in-title' that are associated with a US company, what are the alphabetically first pseudonym and title?
	(8) Among female actresses credited with a voice role who have a registered alternate name (with the alternate name belonging to the same person holding the credit) and played a named character, in movies associated with a US company, what are the alphabetically first alternate name, character name, actress name, and title?
	(9) Among male writers credited in a writing role on Horror or Thriller movies tagged with at least one of 'murder', 'violence', 'blood', 'gore', 'death', 'female-nudity', or 'hospital', that have a vote count on record and are associated with a Lionsgate company—what are the alphabetically first genre, vote count, writer name, and title?
	(10) Among cast members who played a character whose name contains 'man' or 'Man', in titles of kind 'movie' produced after 2000, tagged with at least one of 'superhero', 'marvel-comics', 'based-on-comic', 'tv-special', 'fight', 'violence', 'magnet', 'web', 'claw', or 'laser', and with a complete cast listing—what are the alphabetically first cast member name and title?
krama	*(1) According to NIFC, among the regions that have a corresponding NIFC Geographic Area Coordination Center (GACC), which GACC accumulated the most human-caused wildfire acres from 2020 to 2023 (inclusive)? Use the latest report (by reported date) within each year. Give the GACC abbreviation.*

Table 6 (continued)

Dataset	Query
	(2) According to NIFC, from 2014 to 2024 (inclusive), which US state has the highest total number of lightning-caused wildfires across all NIFC Geographic Area Coordination Centers whose boundaries intersect with that state?
	(3) What is the average number of reported identity thefts (as reported in 2024) across all US metropolitan statistical areas with a 2023 population exceeding one million? Estimate each MSA's 2023 population by linearly interpolating between its 2020 census count and 2024 estimate. Round to 4 decimal places.
	(4) What is the absolute population change (in number of people) from 2020 to 2024 of the metropolitan statistical area with the highest reported identity theft rate per 100K population in the 2024 report?
	(5) From 2012 to 2018 (inclusive), what is the average water temperature (in °C) of the beach that had the most water samples failing to meet bacterial water quality requirements? Round to 2 decimal places.
	(6) From 2008 to 2016 (inclusive), what is the bacterial exceedance rate (as a percentage, rounded to 2 decimal places) of the beach with the lowest average water temperature among all temperature readings collected in that period?
	(7) Among patients recorded in the UCEC_CPTAC3_meta_table_V2.1 sheet, what is the average age of patients whose hyperactivated protein is CDK12? Round to 2 decimal places.
	(8) Among patients recorded in the UCEC_CPTAC3_meta_table_V2.1 sheet with Serous histologic type who were not excluded from the study, what is the hyperactivated protein of the oldest patient?
	(9) What is the mean Kp index, according to the 2024 OMNI2 low-resolution hourly data, for the calendar month in 2024 during which Swarm-B recorded its highest mean thermospheric density? Round to 4 decimal places.
	(10) During the geomagnetic storm period captured in the space-track orbital observations for satellite NORAD 58214, what is the mean BSTAR drag coefficient of satellite NORAD 43180 as recorded in its TLE data? Round to 6 decimal places.
music_brainz_20k	*(1) How much revenue in USD did Apple Music make from Beyoncé's song 'Get Me Bodied' in Canada?*
	(2) Which store earned the most revenue in USD from Brucqe Maginnis' song 'Street Hype' across all countries?
	(3) Which song generated the highest total revenue in USD across all stores and countries?
pancancer_atlas	*(1) For LGG patients, compute the average log10-transformed expression of the IGF2 gene across different histology types. Only include patients with valid IGF2 expression values and histology annotations that are not enclosed in square brackets. Report the final average values with at least four decimal places of precision.*
	(2) Among BRCA patients in the PanCancer Atlas who are alive, which top three histological types show the highest percentage of CDH1 gene mutations?
	(3) Calculate the chi-square statistic to assess the association between histological types and the presence of CDH1 gene mutations in female BRCA patients from the PanCancer Atlas, excluding categories with marginal totals less than or equal to 10, and only focusing on patients with known histological types and consider only reliable mutation entries.
patents	*(1) Identify the CPC technology areas with the highest exponential moving average of patent filings each year (smoothing factor 0.2), and return only the CPC group codes at level 5 whose best year is 2022.*
	(2) Find the CPC technology areas in Germany with the highest exponential moving average of patent filings each year (smoothing factor 0.1) for patents granted in the second half of 2019. Include the full title, CPC group code, and the best year for each CPC group at level 4.
	(3) Which assignees, excluding UNIV CALIFORNIA itself, have cited patents assigned to UNIV CALIFORNIA, and what are the titles of the primary CPC subclasses associated with these citations? Please provide the name of each citing assignee together with the full title of the CPC subclass.
stockindex	*(1) Which stock index in the Asia region has exhibited the highest average intraday volatility since 2020?*
	(2) Among North American stock indices, which indices had more up days than down days in 2018?
	(3) If an investor had made regular monthly investments in all indices since 2000, which 5 indices would have produced the highest overall returns, and what countries do they belong to?
stockmarket	*(1) What was the maximum adjusted closing price in 2020 for The RealReal, Inc.?*
	(2) List all ETF securities listed on NYSE Arca that reached an adjusted closing price above $200 at any point during 2015, and also report the total number of such ETFs.
	(3) List all company names on the NASDAQ-listed Market that were financially troubled (delinquent, deficient, or both) and have trading volume in 2008, for each, report its exisiting non-null average daily trading volume in 2008.
	(4) What are the names (not symbol) of the top 5 non-ETF stocks listed on the New York Stock Exchange (NYSE) that had more up days than down days in 2017? (Up days: closing price > opening price; Down days: closing price < opening price)
	(5) Which 5 companies listed on the NASDAQ Capital Market had the highest number of days in 2019 where the intraday price range exceeded 20% of the low price, list the company names please?
usaspending	*(1) How many distinct contract awards from the Department of Defense have an award amount greater than $1,000,000?*
	(2) How many contract awards went to recipients located in California with an award amount greater than $1,000,000?
	(3) How many distinct recipient UEIs are associated with contracts in NAICS sector 33?
	(4) Among awarding agencies that have issued at least 10 contracts, which agency has the highest proportion of contracts with an award amount greater than $1,000,000? Return the agency name, breaking ties alphabetically.
	(5) How many distinct 2-digit NAICS sectors are represented among contracts with an award amount of at least $10,000,000?

Table 6 (continued)

Dataset	Query
	(6) How many distinct recipients in the recipients database are each associated with more than one UEI?
	(7) Among contracts with an award amount of at least $10,000,000, which 2-digit NAICS sector has the most contracts? Return the 2-digit sector code, breaking ties by the smaller code.
	(8) How many contract awards were made to Lockheed Martin by the Department of Defense with an award amount greater than $1,000,000?
	(9) How many contracts have no English-language description and an award amount greater than $1,000,000?
	(10) Among the top 10 most active contract recipients by total number of contracts, how many of their contracts have a superseded amount entry on record?
yelp	*(1) What is the average rating of all businesses located in Indianapolis, Indiana?*
	(2) Which U.S. state has the highest number of reviews, and what is the average rating of businesses in that state?
	(3) During 2018, how many businesses that received reviews offered either business parking or bike parking?
	(4) Which business category has the largest number of businesses that accept credit card payments, and what is its average rating?
	(5) Which U.S. state has the highest number of businesses that offer WiFi, and what is the average rating for those businesses?
	(6) Which business received the highest average rating between January 1, 2016 and June 30, 2016, and what category does it belong to? Consider only businesses with at least 5 reviews.
	(7) Among users who registered on Yelp in 2016, which 5 business categories have received the most total reviews from those users since 2016?

A.3 Example Database Configuration

The YAML configuration file for the *bookreview* dataset walked through in Section 3.3 specifies its database setup. The agent sees only the logical database names (`books_database`, `review_database`); physical paths and connection details are hidden behind the tools:

```
db_clients:
  books_database:
    db_type: postgres
    db_name: bookreview_db
    sql_file:
    query_dataset/books_info.sql
  review_database:
    db_type: sqlite
    db_path:
    query_dataset/review_query.db
```

A.4 Workload Properties

To show the mapping between the datasets and the workload properties we observed in formative study, we report in Table 7 which queries in each dataset exhibit P2, ill formatted join keys; P3, unstructured text transformation; and P4, domain knowledge. Each cell lists the relevant query IDs, followed by the number of queries in parentheses. We omit P1, integration across heterogeneous databases, from the table because all 104 queries exhibit it.

Section 3.2 already describes and illustrates the four techniques used to introduce P2 and P3. Here, we name these techniques explicitly and show where each is applied:

- T1: Rename a shared identifier across tables, such that the same entity is identified by columns with different names.
- T2: Apply row specific corruptions to join keys, including prefixes, OCR like substitutions, whitespace, and special characters.
- T3: Embed values needed to answer a query within unstructured text and remove the original structured column.
- T4: Express numeric values in varied free text formats.

Some datasets exhibit P2 or P3 "natively," or without a synthetic transformation. Specifically, *agnews* natively requires article classification, *music_brainz_20k* natively requires entity resolution, and *stockindex* natively requires mapping full stock exchange names to their abbreviations. Table 8 maps each dataset to the techniques used to introduce P2 and P3, or marks the property as native.

Table 7: Queries in each dataset that exhibit P2 (ill-formatted join keys), P3 (unstructured text transformation), and P4 (domain knowledge). Each cell lists the query IDs, with the number of such queries in parentheses. P1 (multi-database integration) is omitted because all 104 queries exhibit it.

Dataset	P2	P3	P4
agnews	— (0)	1–4 (4)	— (0)
bookreview	1–3 (3)	1–3 (3)	— (0)
civic_unstructured	1–10 (10)	1–10 (10)	3–6, 10 (5)
crmarenapro	1–13 (13)	1–3, 6–7, 11 (6)	1–3, 6–8, 10–13 (10)
cve	1–10 (10)	1–10 (10)	1–10 (10)
deps_dev_v1	— (0)	1–2 (2)	1–2 (2)
github_repos	— (0)	1–4 (4)	1–4 (4)
googlelocal	— (0)	1–4 (4)	— (0)
imdb	1–10 (10)	— (0)	6, 9–10 (3)
krama	1–10 (10)	1–10 (10)	1–2, 5–10 (8)
music_brainz_20k	— (0)	1–3 (3)	— (0)
pancancer_atlas	— (0)	1–3 (3)	1–3 (3)
patents	— (0)	1–3 (3)	1–3 (3)
stockindex	1–3 (3)	1–3 (3)	1–3 (3)
stockmarket	— (0)	1–5 (5)	1–5 (5)
usaspending	1–10 (10)	1–10 (10)	1–3, 5–8, 10 (8)
yelp	1–7 (7)	1–7 (7)	— (0)
Total	**76/104**	**87/104**	**64/104**

Table 8: Techniques used to introduce P2 (ill-formatted join keys) and P3 (unstructured text transformation) in each dataset. "native" marks datasets that exhibit the property without any synthetic transformation.

Dataset	P2 technique(s)	P3 technique(s)
agnews	—	native
bookreview	T1	T3
civic_unstructured	T1, T2	T3
crmarenapro	T2	T3, T4
cve	T2	T3, T4
deps_dev_v1	—	T3
github_repos	—	T3
googlelocal	—	T3
imdb	T2	—
krama	T1, T2	T3
music_brainz_20k	—	native
pancancer_atlas	—	T3
patents	—	T3
stockindex	native	T3, T4
stockmarket	—	T3
usaspending	T2	T4
yelp	T1	T3

B Tools and Prompts

Table 9 shows the four tools provided to Gemini-3.1-Pro (ReAct).

Below is the system prompt for Gemini-3.1-Pro[3].

```
You are a data analysis agent. Use
    only the tools listed below to
    answer the user's query, based on
    the provided DATABASE DESCRIPTION
    for logical database names and
    their types (SQL or MongoDB), and
    the results of previous tool calls.

TOOLS (system will execute):
- query_db: run a SQL or Mongo query.
    Returns a list of JSON-
```

[3]The system prompts for all models are available at: https://drive.google.com/drive/folders/14SiSz7CVnQA58YOAzFzNjoUu3Wnmfe5t?usp=drive_link

Table 9: Description of tools given to Gemini-3.1-Pro (ReAct).

Tool	Argument(s)	Description	Example
list_db	db_name	Returns the names of all tables in the database specified by db_name, or an error if it does not exist.	list_db('books') returns the names of the tables in the books database: [books_info, authors_info].
query_db	db_name, query	Verifies that query, written in SQL or Mongo, is read-only and, if valid, executes it on the database specified by db_name, returning the query result or an error if execution fails.	query_db('books', 'SELECT title, details FROM books_info;') executes the specified query on the books database and returns the tuples from the books_info table with the specified columns, stored in the variable var_call_123.
execute_python	code	Executes triple-quoted Python code via exec(code, env) in a Docker environment with Python 3.12 and two popular data-processing libraries, Pandas and Pyarrow, preinstalled, with a 600s timeout. Here, env is a dictionary that maps variable names to prior tool call results. The execution result must be JSON-serializable and printed with the prefix line __RESULT__: to be correctly parsed.	execute_python("""import pandas as pd df = pd.DataFrame(var_call_123) res = df[df['details'].str.contains('English', regex=False)].copy() print('__RESULT__:') print(json.dumps(res.to_dict(orient='records'))""") returns the books written in English extracting language information from the free-text details column.
return_answer	answer	Returns answer and terminates agent execution.	The agent terminates and returns the answer using return_answer('The books written in English are Fire Cracker, ...').

```
    serializable records or an error
    string.
  Required args: {"db_name": "<
    logical_db_name>", "query": "<SQL
    or Mongo query>"}
- list_db: list tables or collections
    for a given database.
  Required args: {"db_name": "<
    logical_db_name>"}
- execute_python: run Python code.
  Required args: {"code": "<
    python_code>"}
- return_answer: finish and return the
     final answer (plain text).
  Required args: {"answer": "<final
    plain-text answer>"}

INSTRUCTIONS:
1. After each tool call, its result
    will be stored in a storage under
    a key named after the tool call id
     (you will be told the key name).
    The next message will include both
     the result (or a preview if it's
    large) and the storage key name.
2. Inside execute_python code you may
    read storage entries using the
    provided key names, e.g., if the
    tool call id is 'call-1', you can
    access its result via `locals()['
    var_call-1']` in your code.
3. You cannot modify or reassign those
     storage-provided variables; you
    may read them and create new
    variables as needed.
4. If a tool result is large, the next
     message will include a preview (
    first 10000 characters) and the
    storage entry will be the .json
    file path (a string) where the
    full result is stored. To access
    the full result, your
    execute_python code must open and
    read that .json file.

KEY RULES (must follow exactly):
1. Always use tool calls. Do not
    output plain text, explanations,
    or reasoning.
2. Include all required arguments for
    the tool you call.
3. For query_db, always specify
    db_name and query. Refer the
    DATABASE DESCRIPTION for db_name
    and query format.
4. For PostgreSQL, wrap mixed-case or
    uppercase column names in double
    quotes.
5. For list_db, refer the DATABASE
    DESCRIPTION to specify db_name.
6. Use execute_python for data
    processing as needed.
7. When using execute_python, your
    code will be quoted by triple
    double-quotes and passed as a
    string to `exec(...)` for
    execution in a Python 3.12
    environment with only pandas and
    pyarrow installed. So you must
    ensure your code is compatible
    with this execution method. For
    exampe, do not use triple double-
    quotes in your code, as they may
    interfere with parsing. Do not use
     non-built-in or non-installed
    packages.
8. When using execute_python, your
    code must print the result at the
    end exactly as shown in the PRINT
    FORMAT section below. The printed
    result must be a string that can
    be successfully parsed by json.
    loads() without errors.

PRINT FORMAT (must match exactly):
----BEGIN PRINT FORMAT----
print("__RESULT__:")
print(
    your_json_serializable_string_here
    )
----END PRINT FORMAT----

For simple types (int, float, str,
    bool, None), you may use json.
    dumps() to produce a valid JSON
    string.
For complex or non-JSON-serializable
    types, you must convert them into
```

```
    JSON-compatible forms before
    printing.
For lists or dictionaries, you must
    ensure that all nested elements
    are also converted into JSON-
    serializable types.
9. Return the final answer only via a
    single return_answer tool call. Do
     not include extra text,
    explanation, or formatting.

EXAMPLES:
- query_db:
{"tool": "query_db", "args": {"db_name
    ": "some_db_name", "query": "
    SELECT * FROM some_table LIMIT
    some_limit;"}} # for SQL databases
{"tool": "query_db", "args": {"db_name
    ": "some_db_name", "query": "{"
    collection": "some_collection", "
    filter": {some_filter}, "
    projection": {some_projection}, "
    limit": some_limit}"}} # for
    MongoDB databases

- list_db:
{"tool": "list_db", "args": {"db_name
    ": "some_db_name"}}

- execute_python:
{"tool": "execute_python", "args": {"
    code": "import pandas as pd
# rl1 and rl2 are the keys of two JSON
    -serializable record lists in
    storage
df1 = pd.DataFrame(rl1)
df2 = pd.DataFrame(rl2)
result = pd.merge(df1, df2, on='id').
    head(10).to_json(orient='records')
print('__RESULT__:')
print(result)"}}

- return_answer:
{"tool": "return_answer", "args": {"
    answer": "...final plain-text
    answer..."}}

If you cannot proceed, call
    return_answer with a short
    explanatory message.

Do not output explanations, reasoning,
     or any natural language outside
    of the required tool calls.
```

C Shared Harness Evaluation Results

To further distinguish different models, we evaluate them using a shared ReAct harness (as described in Section 4.1) with GPT-5.2, Claude Opus 4.6, and Gemini 3.1 Pro. Due to budget limits, we evaluated about half of the queries; 54 queries across 12 datasets (selecting the least expensive queries), with five trials per model per query. Results are reported in Table 10.

Table 10: Per-dataset pass@1 (P@1) and pass@5 (P@5) under the shared ReAct harness, over 54 queries across 12 datasets with five trials per model per query. **Green**: highest per dataset and metric; 0: zero score.

	GPT-5.2		Claude-Opus-4.6		Gemini-3.1-Pro	
Dataset	**P@1**	**P@5**	**P@1**	**P@5**	**P@1**	**P@5**
agnews	0	0	**0.30**	**0.50**	0.25	**0.50**
bookreview	0.53	0.67	**1.00**	**1.00**	0.93	**1.00**
crmarenapro	0.52	0.77	**0.79**	**0.92**	0.75	0.85
deps_dev_v1	0	0	0.40	**0.50**	**0.50**	**0.50**
github_repos	0.15	0.25	0.35	**0.50**	**0.40**	**0.50**
googlelocal	0.30	0.50	0.75	0.75	**0.80**	**1.00**
music_brainz_20k	0.07	0.33	0.40	0.67	**0.73**	**1.00**
pancancer_atlas	0.40	**0.67**	0.47	**0.67**	**0.53**	**0.67**
patents	0	0	0	0	0	0
stockindex	**0.93**	**1.00**	0.73	**1.00**	**0.93**	**1.00**
stockmarket	0.32	0.80	0.72	0.80	**0.80**	**1.00**
yelp	0.20	0.29	**0.66**	**1.00**	0.63	**1.00**
Stratified avg	0.29	0.44	0.55	0.69	**0.60**	**0.75**

Table 11: Number of exploratory tool calls (e.g., schema lookups or SELECT * LIMIT 5) per trial, with share of all tool calls in parentheses, averaged over one trial per query. A Claude Code subagent classifies each call.

Setting	**Claude-Sonnet-4.6**	**GPT-5.5**	**Gemini-3.1-Pro**
full info	14.5 (39.2%)	7.1 (50.9%)	6.9 (21.8%)
w/o hints	15.0 (45.8%)	6.2 (45.0%)	7.3 (23.6%)
w/o desc + hints	21.1 (55.1%)	10.4 (60.6%)	14.5 (40.9%)

D Ablation Results

Table 12 and Table 13 report per-dataset average pass@1 accuracy when agents run without hints ("w/o hints") and without both hints and description files ("w/o desc + hints"), respectively. Table 14 and Table 15 report per-dataset cost (in USD) of "w/o hints" and "w/o desc + hints", respectively. Table 11 reports the number of exploratory tool calls per trial in each setting.

E Failure Description and Examples

E.1 Failure-Mode Frequency Plots

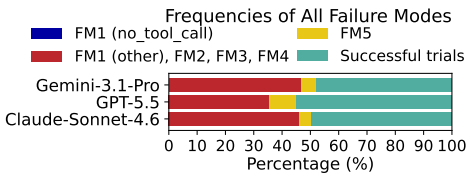


Figure 5: Failure-mode breakdown across all 1,560 trajectories. Runtime errors (**FM5**) are rare; **FM1 (no_tool_call)** only occurs in one Gemini-3.1-Pro trial.

Table 12: Pass@1 score by dataset when agents run without hints ("w/o hints"), with the average across all 17 datasets. One dataset remains completely unsolved by every agent. **Green**: highest per dataset; 0: zero Pass@1; *Gray*: partial results, where some queries in the dataset have not yet been evaluated.

Dataset	Claude-Sonnet-4.6	GPT-5.5	Gemini-3.1-Pro
agnews	*0.35*	0.45	0.30
bookreview	0.80	**1.00**	**1.00**
civic_unstructured	**0.50**	0.48	0.48
crmarenapro	0.69	0.75	**0.77**
cve	*0*	**0.18**	0.08
deps_dev_v1	0.30	**0.40**	**0.40**
github_repos	**0.45**	0.35	0.30
googlelocal	0.70	**1.00**	0.70
imdb	0.58	**0.64**	0.36
krama	0.56	**0.74**	0.16
music_brainz_20k	0	**0.27**	**0.27**
pancancer_atlas	**0.67**	0.60	0.60
patents	0	0	0
stockindex	0.87	0.80	**1.00**
stockmarket	0.92	**0.96**	0.72
usaspending	0.40	0.38	**0.42**
yelp	0.66	**0.91**	0.66
Average	0.50	**0.58**	0.48

Table 13: Pass@1 score by dataset when agents run without both hints and description files ("w/o desc + hints"), with the average across all 17 datasets. One dataset remains completely unsolved by every agent. **Green**: highest per dataset; 0: zero Pass@1; *Gray*: partial results, where some queries in the dataset have not yet been evaluated.

Dataset	Claude-Sonnet-4.6	GPT-5.5	Gemini-3.1-Pro
agnews	*0.10*	0.30	0.30
bookreview	0.73	**1.00**	**1.00**
civic_unstructured	0.44	**0.52**	0.32
crmarenapro	0.68	**0.82**	0.80
cve	*0.06*	**0.22**	0.12
deps_dev_v1	0.20	**0.50**	0.40
github_repos	**0.50**	0.30	0.40
googlelocal	0.75	**0.80**	0.70
imdb	*0.40*	**0.56**	0.42
krama	*0.16*	**0.68**	0.14
music_brainz_20k	0.07	**0.27**	0.20
pancancer_atlas	**0.67**	0.53	0.47
patents	**0.07**	0	0
stockindex	**0.93**	**0.93**	**0.93**
stockmarket	**0.92**	0.84	0.72
usaspending	*0*	0.30	**0.32**
yelp	0.71	**0.83**	0.60
Average	0.43	**0.55**	0.47

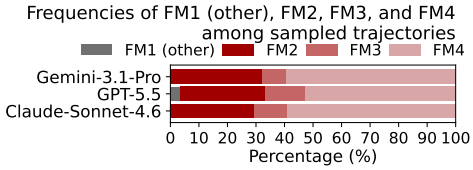


Figure 6: Decomposition of the LLM-judged red bar in Figure 5 over 196 trajectories. **FM4** and **FM2** account for 90% of failures.

Table 14: Total cost (USD) per dataset when agents run without hints ("w/o hints"). **Green**: cheapest per dataset; *Gray*: estimated total cost for datasets with partial results.

Dataset	Claude-Sonnet-4.6	GPT-5.5	Gemini-3.1-Pro
agnews	*23.8*	34.7	10.7
bookreview	3.7	3.0	1.8
civic_unstructured	108.6	59.9	26.9
crmarenapro	26.1	23.3	16.9
cve	*273.4*	129.0	94.7
deps_dev_v1	3.5	5.3	2.8
github_repos	4.4	9.7	4.7
googlelocal	3.8	3.9	1.6
imdb	46.8	80.6	23.9
krama	90.9	58.1	49.8
music_brainz_20k	0.9	1.8	1.3
pancancer_atlas	5.6	5.9	3.0
patents	8.5	11.9	4.9
stockindex	2.7	3.2	2.6
stockmarket	3.4	3.3	9.2
usaspending	54.0	36.9	16.4
yelp	10.2	11.8	8.3
Total	670.4	482.5	279.5

Table 15: Total cost (USD) per dataset when agents run without both hints and description files ("w/o desc + hints"). **Green**: cheapest per dataset; *Gray*: estimated total cost for datasets with partial results.

Dataset	Claude-Sonnet-4.6	GPT-5.5	Gemini-3.1-Pro
agnews	*26.4*	46.7	13.4
bookreview	3.8	4.4	3.3
civic_unstructured	111.0	52.9	31.5
crmarenapro	29.7	32.4	16.7
cve	*204.6*	79.4	69.2
deps_dev_v1	3.9	9.4	3.4
github_repos	6.9	15.0	7.2
googlelocal	5.2	4.7	2.0
imdb	*62.4*	59.0	25.4
krama	*107.2*	62.2	70.9
music_brainz_20k	1.3	2.7	0.7
pancancer_atlas	6.0	7.4	3.0
patents	11.9	14.7	6.1
stockindex	3.1	4.7	3.2
stockmarket	5.6	6.1	15.6
usaspending	*84.7*	31.1	19.8
yelp	12.3	20.3	7.9
Total	686.0	453.0	299.2

E.2 FM Annotation Prompt

Below is the prompt template for GPT-5 to annotate **FM1 (other)**, **FM2**, **FM3**, and **FM4** for a failed trajectory.

```
You are given a trace of a FAILED task executed by a data agent, along with the task query and the ground-truth answer. Your task is to diagnose WHY the agent's final answer does not match the ground truth using the failure modes FM1-4 defined below.

The agent had access to the following tools:
- list_db: list tables in a database
- query_db: execute SQL queries
- execute_python: run Python code for data processing
- return_answer: terminate and return the final answer

The trace records the complete interaction between the data agent and the available tools. That is, the trace contains:
- Exploratory tool calls (trial-and-error)
- Tool calls that directly contributed to the final answer

## IMPORTANT INSTRUCTIONS:
1. First, identify ONLY the tool calls whose outputs were used (directly or indirectly) to produce the final answer. Ignore abandoned, failed, or exploratory tool calls that did not affect the final output.
2. Base your failure analysis ONLY on those contributing tool calls.
3. You must mark at least one failure mode as "yes". By default, select exactly one failure mode-the most specific one that best explains why the final answer is incorrect. Select multiple failure modes only if the failure cannot be adequately explained by any single mode alone (i.e., each selected mode captures a distinct, necessary cause visible in the contributing tool calls).
4. Only mark a failure mode as "yes" if you can point to a concrete example in the trace.
5. Provide a single-sentence summary explaining the failure, explicitly referencing the trace behavior.

## FAILURE MODELS
FM1 - Fails Before Planning
Definition:
- The agent does not attempt to solve the query.
Example:
- The agent does not issue any tool calls.

FM2 - Incorrect Plan
Definition:
- The agent attempts to solve the user's query, but the PLAN (i.e., the logical structure) of the solution is wrong. That is, even if all steps were executed perfectly (e.g., correct data selection, correct implementation, no execution errors), the plan cannot produce the ground-truth answer.
Examples:
- Missing operations specified or implied by the user.
- Adding constraints not requested by the user.
- Stops early and returns an answer before all requirements of the query are completed.

FM3 - Correct Plan, Wrong Data Selection
Definition:
- The agent follows a theoretically correct plan for answering the user's query, but selects incorrect data sources in its implementation, such that the required information exists but is retrieved from the wrong database, table, collection, column, or field.
Example:
- Using an incorrect column in a selection or filtering condition (e.g., a WHERE clause references a column that does not represent the queried attribute, even though the correct column exists elsewhere in the schema).
- Querying or joining a table that does not represent the queried entity, despite the correct table exists elsewhere in the schema.

FM4 - Correct Plan and Data Selection, Incorrect Implementation
Definition:
The agent the correct plan and selects the correct tables and columns, but implements the computation incorrectly.
Examples:
- Arithmetic errors, such as computing an aggregate with an incorrect formula (e.g., dividing by the number of unique entities rather than the total number of contributing entities or records when calculating an overall average).
- Incorrect regular expressions or parsing rules, such that extracted values include unintended matches or miss required values (e.g., patterns that capture unrelated numbers or fail to match the intended tokens).
```

```
- Parsing errors, such as using unsupported syntax, improper escaping, or assumptions about the execution environment that cause incorrect behavior or failed execution.
- Incorrect join implementation, where the agent applies an invalid join condition or fails to normalize or transform identifiers before joining, even though the trace indicates awareness that normalization or alignment is required.

## QUERY
{{uesr_query}}

## GROUND-TRUTH ANSWER
{{ground_truth_answer}}

## FAILED TRACE
{{failed_trajectory}}

## OUTPUT FORMAT
Your output MUST follow this exact format, starting after @@ and ending before @@:

@@
A. Freeform text summary of the failure:
<one sentence>
B. Failure modes encountered:
FM1: <yes or no>
FM2: <yes or no>
FM3: <yes or no>
FM4: <yes or no>
@@
```

E.3 Examples of Failed Trajectories

Examples in this section are drawn from the bookreview dataset, with the following description and hints:

```
DATABASE DESCRIPTION:
You are working with two databases to solve this query.

Here are the descriptions of these two databases:

1. books_database
   - This database is stored in a PostgreSQL database and contains Amazon book information including descriptions, price, details, title, etc. up to 2023.
   - This database consists of one table:
     - books_info:
       - Fields:
         - title (str): Book title
         - subtitle (str): Book subtitle
         - author (str): Book author(s)
         - rating_number (int): Total number of ratings received
         - features (str): Book features (stored as string representation of list/dict)
         - description (str): Book description (stored as string representation of list/dict)
         - price (float): Book price
         - store (str): Store information
         - categories (str): Book categories (stored as string representation of list/dict)
         - details (str): Additional book details
         - book_id (str): Unique book identifier

2. review_database
   - This database is stored in a SQLite database and contains Amazon book review information including ratings, text, helpfulness votes, etc. up to 2023.

   - This database consists of one table:
     - review
       - Fields:
         - rating (float): Rating given by reviewer (1.0-5.0 scale)
         - title (str): Review Title
         - text (str): Review text content
         - purchase_id (str): Unique identifier linking to book_id in books_info table in books_database
         - review_time (str): Timestamp when review was posted
         - helpful_vote (int): Number of helpful votes received
         - verified_purchase (bool): Whether purchase was verified
```

```
HINTS:
- In books_info, the "description", "categories", and "features" content appears to be in list or dictionary format, but they are actually stored as strings in the .sql file.
- The fields "book_id" in books_info and "purchase_id" in review refer to the same book entities across different tables. While the field names are not exact matches, they can be joined using a fuzzy join approach to solve the query.
- For some queries, you could get needed information from "categories" or "details" columns in books_info.
```

There are three queries under bookreview, denoted as Q1, Q2, and Q3. We list the queries and their corresponding ground-truth answers A1, A2, and A3 below:

(Q1) *Which decade of publication (e.g., 1980s) has the highest average rating among decades with at least 10 distinct books that have been rated? Return the decade with the highest average rating.*
(A1) 2020
(Q2) *Which English-language books in the 'Literature & Fiction' category have a perfect average rating of 5.0? Return all matching books.*
(A2) The Sludge; Something That Feels Like Truth (Switchgrass Books); Kennebago Moments; Hollywood Confessions: Hollywood Headlines Book #3 (Hollywood Headlines Mysteries); Forged in Blood (Freehold); Local Honey; "Exits, Desires, & Slow Fires"; Fire Cracker; Reunion: The Children of Lauderdale Park; Childe Harold of Dysna; The Prophet: With Original 1923 Illustrations by the Author; Knowing When To Die: Uncollected Stories; Liza of Lambeth; Child Of The King A Journey of Hope Book 1: Earthly Story With A Heavenly Message; The Melancholy Strumpet Master;
(Q3) *Which books categorized as 'Children's Books' have received an average rating of at least 4.5 based on reviews from 2020 onwards?*
(A3) Around the World Mazes; Behind the Wheel (Choose Your Own Adventure #35)(Paperback/Revised); Benny Goes To The Moon: The great new book from Top Children's entertainer Gerry Ogilvie (1); "Cheer Up, Ben Franklin! (Young Historians)"; Favorite Thorton W. Burgess Stories: 6 Books; Egypt (Enchantment of the World); "Pokemon: Sun & Moon, Vol. 8 (8)"; The Library Book; LunaLu the Llamacorn; Monstrous Stories #4: The Day the Mice Stood Still; The Old Man and the Pirate Princess; Trouble in the CTC!: The Terra Prime Adventures Book 2; "Clark the Shark: Tooth Trouble, No. 1"; Cleo Porter and the Body Electric;

For brevity, we analyze only the failure modes of selected example trajectories and omit the full interaction traces[4].

E.3.1 FM2: Incorrect Plan for Averaging

Query. Q1
Analysis. This trajectory fails due to an incorrect averaging plan: it computes decade-level averages by averaging per-book average ratings, whereas the query requires directly averaging all ratings within each decade. Specifically, the final answer in Line 423 derives from the `execute_python` tool call in Line 368, which uses results from prior `query_db` calls that retrieve records from `books` (Line 48; `var_call_mC9eh9kdqR7TFrzmoKhf7oa0`) and `reviews` (Line 36; `var_call_clwW1HpxqxlCKDXJvn9Iim9W`). The agent first computes per-book average ratings—treating each `purchase_id` as a book identifier—when processing review records (Line 38). After merging the retrieved `books` and `reviews` tables in Python (Line 396), it computes decade-level averages by averaging these per-book averages (Line 401), resulting in an incorrect aggregation.

E.3.2 FM2: Missing Operations

Query. Q2
Analysis. This trajectory fails by missing a required constraint: the selected books must have an average rating of exactly 5.0. Specifically, the final answer in Line 591 is produced by the `execute_python` tool call in Line 561, which depends on a prior `execute_python` call in Line 525, itself derived from a `query_db` call in Line 113. All three calls contributing to the final answer ignore the constraint that the average rating must equal 5.0.

E.3.3 FM2: Adding Operations

Query. Q3
Analysis. This trajectory fails by introducing unwarranted operations: it adds `LIMIT 200` when retrieving review records (Line 38) and `LIMIT 500` when retrieving book records (Line 50), while the query does not require any such limits.

E.3.4 FM3: Incorrect Column Selection

Query. Q2
Analysis. This trajectory fails by selecting an incorrect column. To filter books written in English, the agent relies on columns other than the correct one (i.e., `details`) in the `books_info` table, although the database description already indicates `details` as the correct column (Line 39). Specifically, in Line 125 of the trajectory below, the agent attempts to search for "English" using `details`, `description`, and other columns. However, the book records it inspects originate from `var_call_he4GeOzFNhdoBfpZPxkmX09E` (Line 110), which is produced by the `query_db` call in Line 51 executing `SELECT book_id, title, author, categories, description`

[4]Complete trajectories are available at https://drive.google.com/drive/folders/1FES67CWgfOaR-1FRcqoIeyvNaUV4NFZ6?usp=drive_link. In these example trajectories, for presentation clarity, we omit system and user prompts, and truncate large tool results to 300 characters.

`FROM books_info;` This query does not retrieve the `details` column. In effect, the agent searches for "English" using columns other than the correct one (`details`), leading to failure.

E.3.5 FM4: Incorrect Regular Expression

Query. Q1

Analysis. This trajectory fails due to an overly permissive regular expression. The agent uses a pattern to extract any four-digit string starting with 19 or 20 (Line 66) and then selects the smallest extracted value (Line 73) as the publication year. This approach may incorrectly extract `1932` from an ISBN rather than the true publication year, `2004`, as in the example shown in Line 17.