ARGO-BENCH: EVALUATING DATA AGENTS ON ENTERPRISE-SCALE WORKFLOWS

Anonymous authors
Paper under double-blind review

ABSTRACT

Real-world enterprise data science and analytics workflows typically require reasoning across dozens of tables, performing statistical analysis and intermediate computations, and taking actions based on findings. Established benchmarks do not represent realistic, full-scale enterprise data warehouses due to the highly sensitive and confidential nature of such data. We introduce Argo-Bench, an evaluation framework comprising 210 complex data science and analytics tasks in a simulated environment. We simulate a terabyte-scale consumer food delivery platform in New York City with grounded economics and well-known patterns ranging from fraud scenarios to marketplace financial incentives based on public data sources, peer-reviewed industry literature, and regulatory data disclosures and filings. We then export this world to an ERP warehouse modeled on the Oracle E-Business Suite schema with 235 tables and 7.49 billion rows. By withholding the simulator's ground-truth state from the warehouse projection exposed to the evaluated agent, we can design realistic tasks that require reconstructing facts through the warehouse's idiosyncrasies before acting on them. Every task has an executable reference solution that demonstrates solvability using only the warehouse projection. Argo-Bench goes beyond text-to-SQL, requiring the agent to file actions such as banning users or discontinuing promotions. The strongest of 12 frontier and open-weight models reaches a score of at least 95 out of 100 on only 32.9% of tasks and averages 59.3 points. We propose that progress on Argo-Bench demonstrates meaningful progress towards understanding, navigating, and taking action within real-world data environments.

1 INTRODUCTION

In recent years, data agents have advanced from answering simple text-to-SQL questions over small, well-documented schemas to performing long-horizon data science tasks, publishing durable company-wide assets such as dashboards, and acting on their findings, for instance, by adjusting promotions and banning fraudulent accounts (Sun et al., 2025; Liu et al., 2026; Li et al., 2026). Rising scores on established data benchmarks could be read as a sign that enterprise data work is close to solved. The best Spider 2.0-Snow score has risen from 23.8% at release (Lei et al., 2025) to 96.7%, and the top BIRD entry reaches 82.4% against a human 93.0%.[1]

However, current benchmarks primarily evaluate text-to-SQL performance and are not representative of enterprise-grade agentic data workflows in three important ways. First, established benchmarks are not enterprise-shaped. Rather, they are built from public data. Over two thirds of Spider 2.0-Snow's questions run on BigQuery public datasets, and another quarter on local sample databases (Lei et al., 2025), sources whose schemas are documented in tutorials and textbooks. Their databases are collections rather than systems. By our analysis, at least two thirds of Spider 2.0-Snow's tables are date-, geography-, or version-sharded copies of another table, and three quarters of its questions with public gold SQL read at most two logical tables (Appendix H). In an enterprise warehouse, by contrast, different modules must reconcile to the same numbers, and a single business event may touch ten or more tables (Plattner, 2014).

[1] Spider 2.0 leaderboard: `https://spider2-sql.github.io/`; BIRD leaderboard: `https://bird-bench.github.io/`; both accessed September 24, 2026.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

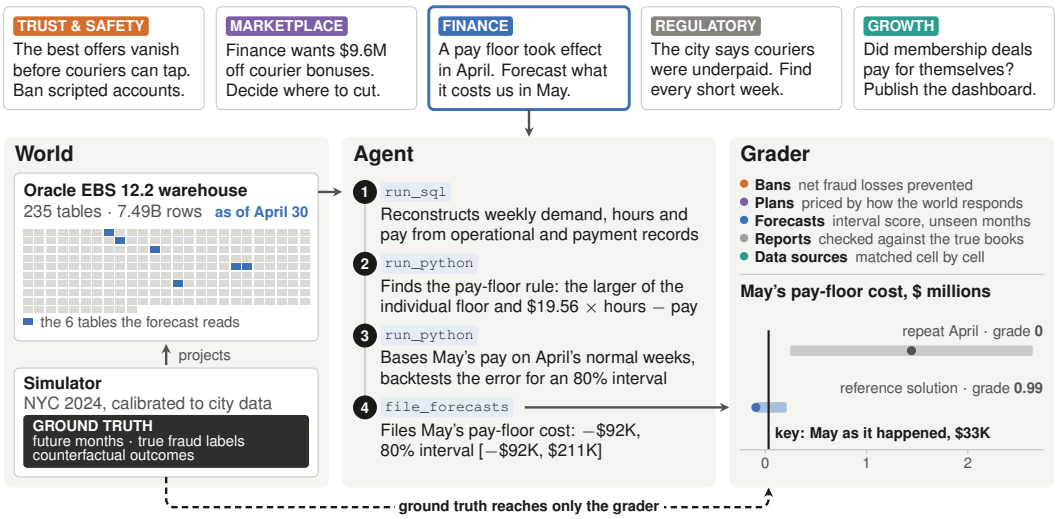

Figure 1: Argo-Bench grades the filings of an agent with an incomplete view of the world. Top: one task from each of the five business areas, colored by what the agent files (as in Figure 3). Bottom: the forecasting task fc-12 (Appendix L). The simulator projects a view of the world into an Oracle EBS warehouse up to April 30, omitting information. The reference solution reads 6 of the 235 tables, recovers the pay-floor rule, and files a forecast for May with an 80% interval. Only the grader sees May as it happened. Repeating April's $1.44 million misses by $1.41 million and grades 0, while the reference solution grades 0.99.

Second, they are not end-to-end. Many of the most valuable enterprise data science tasks cannot be written as a SQL query, since a demand forecast must be fitted, a fraud model trained, and a courier schedule optimized with statistical, machine-learning, and optimization libraries. The work does not end with an answer, either. A finding becomes a decision, which is ultimately assessed by its return on investment. For instance, a promotion with strong redemptions can still lose money if most of those orders would have been placed anyway (Xu et al., 2025b). Checking a result against a gold answer cannot distinguish two such decisions by their payoff.

Third, they are not verifiable. On real data, a benchmark can only measure agreement with its annotators, since writing the answer key requires solving the task. Even where the answer is in the data, annotators often miss it. In a recent audit, 62.8% of Spider 2.0-Snow's released gold queries and 52.8% of BIRD Mini-Dev's were found to be erroneous, most often because annotators misread the data or the schema, and correcting BIRD's errors moved agents by up to nine places on its leaderboard (Jin et al., 2026). For many of the most valuable enterprise tasks, the answer may not be in the data at all. Fraud that was never detected leaves no label (Altman et al., 2023), and the outcome of a decision that was never taken cannot be observed.

Large enterprises keep the records that financial planning, forecasting, and fraud detection require in enterprise resource planning (ERP) systems (Davenport, 1998) such as Oracle E-Business Suite (EBS), SAP S/4HANA, and Oracle Fusion Cloud, extended with custom tables representing idiosyncrasies and workflows unique to the business (Brehm et al., 2001). Because the same systems hold the company's ledgers, payroll, and customer records, access to them is heavily restricted. For this reason, ERP data has remained practically unexplored in benchmarks to date. When such data is released, it must be anonymized, which can break the relational structure that answers depend on. In an early release of BEAVER (Chen et al., 2024), the closest attempt to date, Chung et al. (2025) report primary keys that violate their uniqueness constraints and questions whose gold query is null.

We instead simulate a business and grade against the simulation ground truth (Figure 1). Argo-Bench models a food delivery platform in New York City in 2024, a three-sided marketplace whose economics are disclosed to the city every month (DCWP, 2024a) and which absorbed a real minimum-pay increase in April 2024 (DCWP, 2024b). The simulator is calibrated to these disclosures and to public filings, and none of its data is generated by a language model. We project this world into an Oracle EBS warehouse that omits the simulator's latent state, so tasks are harder to solve than to

verify (Song et al., 2025). The agent must reconstruct facts from the warehouse, while the grader reads them off the state. Agents work in a sandboxed Python environment with statistics, machine-learning, and optimization libraries (Appendix F), and file their decisions and results through a mission-control interface rather than returning a query. Because the grader knows the latent state, it can grade a decision by its consequences. For instance, a list of banned accounts is scored by the fraud losses it prevents, including losses from fraud that the platform never detected, net of the revenue lost from wrongly banned customers. Another advantage of computing answer keys from latent state is that a wrong key is a bug that can be fixed, rather than a judgment up for debate.

Simulated environments are an established practice in a broad range of domains (Altman et al., 2023; Lopez-Rojas et al., 2016; Trivedi et al., 2024; Barres et al., 2025; Xu et al., 2025a; Huang et al., 2026), and synthetic warehouses have long been used to benchmark data systems (Nambiar & Poess, 2006; Ghazal et al., 2013). Even Spider 2.0 draws a fifth of its questions from synthetic, obfuscated, or textbook sample schemas (Appendix H). To our knowledge, what prior simulated benchmarks lack is an enterprise-scale warehouse requiring global coherence across a large set of tables, and a grader that uses the simulator's latent state as ground truth to score the consequences of the agent's actions.

We evaluate 12 frontier and open-weight models on Argo-Bench. The strongest, Claude Opus 5.5, solves 32.9% of tasks and averages 59.3 points, and nine of the twelve models average below 35 points. Models often carry out sophisticated analyses of the wrong quantity or toward the wrong objective.

We make the following contributions:

- A synthetic, terabyte-scale public ERP dataset in Oracle EBS format for a 2024 New York City food delivery platform, comprising 235 mutually constraining tables, 81 million orders, 3.4 million active customers, and a total of 7.49 billion rows, in which an order resolves into dispatch decisions, courier pay, merchant payouts, and balanced general-ledger journals.
- A grader that scores the consequences of an action rather than the correctness of a query, using the simulator's latent state as ground truth, for instance to score bans by the fraud losses they prevent and forecasts against held-out months, supporting tasks across fraud detection, forecasting, and financial planning.
- A benchmark of 210 such tasks, from publishing a dashboard data source to fitting forecasts and banning fraudulent accounts, on which the strongest model solves 32.9% of tasks and averages 59.3 points.

Argo-Bench is public. The dataset is released on Hugging Face under CC BY 4.0,[2] and the tasks, reference agent, tool server, and sandboxes under the Apache License 2.0.[3] A second world generated from a private seed, with different customers, couriers, and answer keys, is held out: the public leaderboard is scored only on this world, so a score on Argo-Bench cannot be earned by training on or memorizing the public warehouse. A demo at `https://argo-bench.org` lets readers browse the orders and deliveries of a world at 10% scale.[4]

2 Benchmark Construction

2.1 World Simulation

We simulate a food delivery platform in New York City (NYC) in 2024, similar to DoorDash, Grubhub, or Uber Eats. What is public about such platforms is aggregate: the quarterly reports the city publishes and the platforms' own filings give totals, but no record of an order, customer, or courier can be released without exposing the platform's customers, couriers, and margins. The world is therefore built from roughly fifty public datasets and reports, each lending one mechanism (Appendix A). Uber and Lyft trips give the time to drive between two zones at a given hour, for

[2] `https://huggingface.co/datasets/DoubleBlindAnon912/iclr-double-blind-dataset`

[3] `https://anonymous.4open.science/r/Argo-Bench-8B75/`

[4] All links are anonymized and do not reveal the authors' identities.

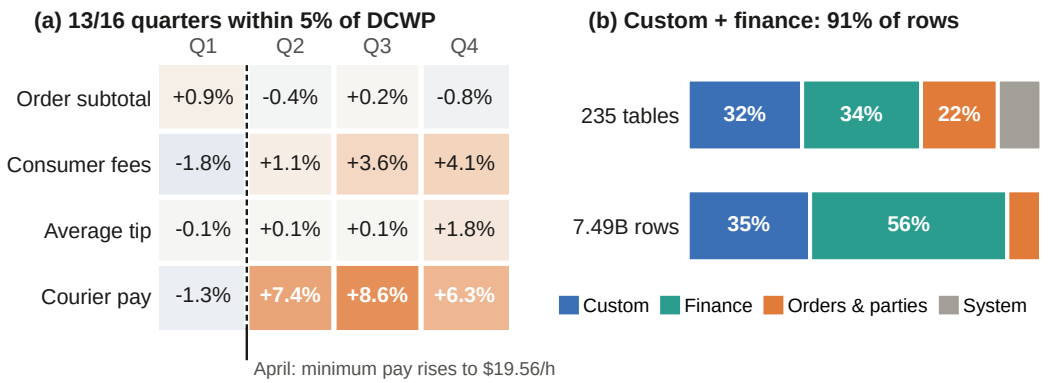

Figure 2: The simulated world matches the city's reported economics, and its warehouse is ERP-shaped. (a) Deviation of per-delivery economics from the 2024 NYC DCWP quarterly anchors. Courier pay runs 6–9% above the anchor after the April minimum-pay increase, and all other figures stay within 5%. (b) Shares of the warehouse's 235 tables and 7.49 billion rows by EBS module group. Custom extensions alone hold 32% of the tables and 35% of the rows.

instance, and grocery orders give the days between a customer's orders. Donors play one of three roles. *Identity* donors are public records of real things in the city, such as its 45,834 restaurants, 1.07 million addresses, and 260 taxi zones, and enter the world as they are. *Shape* donors are measured elsewhere, on other people, in another city, or in another year, and lend only a distribution. *Anchors* are the published totals, which the world is calibrated to reproduce and never samples from. No record in the world is generated by a language model.

We chose food delivery in NYC because its economics are unusually well documented. Delivery apps must report their monthly orders, consumer spending, merchant fees, and courier earnings, productivity, and hours worked to the NYC Department of Consumer and Worker Protection (DCWP), which publishes them quarterly (DCWP, 2024a), and the 10-K filings of DoorDash and Grubhub give the shape of a platform's balance sheet (DoorDash, Inc., 2024b; 2025; Grubhub Inc., 2021). Following Walonoski et al. (2018), we calibrate the world to these anchors, targeting a 61% market share to model a dominant platform. The result has 81 million orders and 3.4 million active customers, and it must balance incentives on three sides: quests and suggested pay for couriers, promotions and surge pricing for customers, and co-funded campaigns for merchants. Its per-delivery economics stay within 5% of the DCWP's figures in 13 of 16 quarterly comparisons (Figure 2a).

We selected 2024 because it contains a real extrinsic shock to these economics. NYC began enforcing a minimum pay rate of $17.96 per hour before tips for app-based restaurant delivery workers in December 2023 and raised it to $19.56 on April 1, 2024 (DCWP, 2024b). We model the platforms' response with a new courier scheduler that activates on that date, after which courier pay runs 6–9% above the anchor (Figure 2a). The levers our platform uses may differ from those of the real platforms, but the aggregate effect matches, and because the world absorbs the same shock, we can ask realistic forecasting questions about it (Appendix C).

Additionally, we insert fraud patterns for which there is public evidence that they are a major issue for delivery platforms: couriers who pick up orders and never deliver them (Al Jazeera, 2025), refund collusion between storefronts or couriers and their regular customers (DoorDash, Inc., 2023), promotion farming by rings of new accounts (DoorDash, Inc., 2025; Incognia, 2025), account takeovers and bust-outs with stolen cards (DoorDash, Inc., 2025; Whittaker, 2018), GPS spoofing (Incognia, 2022) and offer-grabbing bots (Chapman & Mehrotra, 2020), rented courier accounts (DoorDash, Inc., 2024a), shell storefronts (Heier, 2023), and payouts diverted to changed bank accounts (Maycock, 2024). Each pattern is calibrated both to how separable real card fraud is and to how often honest customers share a device, an address, or a card, since the latter sets a detector's precision (Appendix A). The simulator generates the world from the donors, calibrates it to the anchors, inserts the fraud, and projects the result to the EBS format (Section 2.2).

2.2 Warehouse Design and Validation

The simulator's last step projects the world into what an analyst actually sees, the analytics export of a greenfield Oracle E-Business Suite (EBS) 12.2 instance. We chose EBS because its data model is publicly documented, so the realism of our schema can be measured against a reference rather than asserted. We designed the schema with three ERP consultants with 14 to 31 years of experience in the domain.

Every standard table and column in our warehouse exists in the data dictionary of Oracle's EBS 12.2 Vision instance, the demo environment Oracle provides as a reference. The converse does not hold, and should not. Our tables carry 52% of the columns of their Vision counterparts. Because EBS tables are wide enough to serve any industry, analytics exports routinely omit the columns a business does not use. As in most deployments, the standard modules are also extended, with 76 of the 235 tables (32%) custom and holding 2.59 billion of the 7.49 billion rows (Figure 2b and Appendix B). Courier incentives, dispatch decisions, and delivery delays live only in these tables. The difficulty lies in the links between canonical and custom tables rather than in the EBS schema itself.

What the warehouse lacks is history. It has no deprecated tables that overlap active ones in purpose, and its accounting reconciles exactly, whereas long-lived warehouses accumulate both over years of migrations and acquisitions (Vogelsgesang et al., 2018). The consultants named this the most significant difference from their customers' systems, and we keep it deliberately, for two reasons. First, it keeps the ground truth unambiguous. When a legacy table and an active one disagree, the correct answer depends on conventions known only to the data team, and the grader would have to encode them. Second, mature warehouses compensate for their idiosyncrasies with semantic layers, data models, and institutional knowledge (Kandel et al., 2012). Providing these would test how well the agent uses a curated layer rather than whether it understands how an enterprise organizes its data. Withholding them from a messy warehouse would make tasks ambiguous rather than hard. Clean does not mean small. Workloads span hundreds of tables, yet their queries touch only a few distinct sets of them (van Renen et al., 2024). We argue that deciding how much of the warehouse to explore is itself the skill under test, and a greenfield warehouse isolates it.

Otherwise, the consultants found the schema realistic, with two further omissions. It records no foreign-currency transactions, since no donor dataset covers the currencies visitors pay in, and it has only 43 balance sheet accounts (Section 5).

2.3 Task Design

Argo-Bench contains 210 tasks in five business areas, named for the teams of a delivery platform that would own them (Figure 3a). Trust and safety tasks are enforcement, where the agent finds fraud and abuse and acts on the accounts involved. FP&A tasks forecast unit economics and rebuild finance dashboards, marketplace tasks forecast courier supply and allocate budgets such as courier bonuses, accounting tasks report final values after the fact, and growth tasks measure, forecast, and publish the results of promotions and memberships.

Each task has four parts (Appendix G). The *prompt* states the problem as a stakeholder would, for instance "Work out which couriers are stealing food and ban them for reason 'too many missing orders' carrying their `VENDOR_ID`." The *scope* sets the last month of 2024 visible to the agent. The *expectations* list the filings the grader requires, each with its action, keys, and grading mode. The *answer key* is frozen before any run and comes from SQL over the latent tables or from the simulator's own labels, such as which couriers stole orders. Because the world is simulated, these labels are exact and need no anonymization (Gadotti et al., 2024).

Unlike most data science and analytics benchmarks, Argo-Bench does not ask agents to return a query. Agents file actions to a mission-control interface through a Python library in their sandbox (Appendix E). This design has four advantages. First, it admits advanced data science tasks, since many workflows require machine learning, operations research, and mathematical optimization libraries to complete. Second, it supports end-to-end workflows. Emitting the correct SQL is not enough in practice, as a task may only be finished by discontinuing an unprofitable campaign, banning a set of users who are likely committing fraud, or holding the payouts of a suspicious merchant. Third, filings are explicit declarations of intent. When benchmarks compare SQL results (Li et al., 2023; Lei et al., 2025; Chen et al., 2024), it is hard to determine whether a close number is the

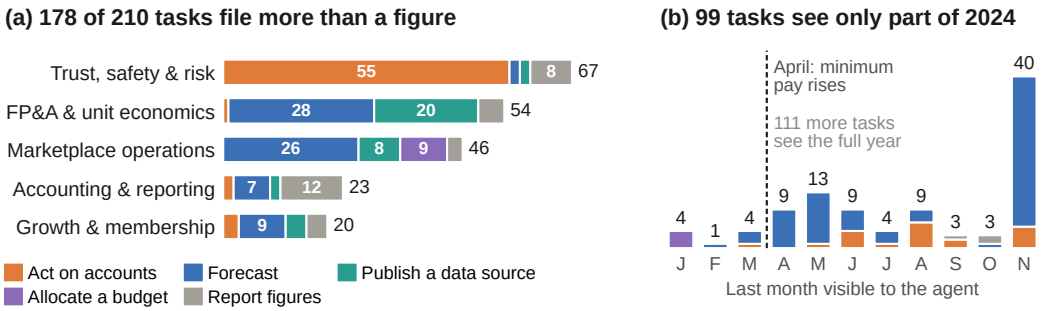

Figure 3: Most tasks end in a decision rather than a number. (a) Tasks per business area, by what the agent files. Multi-part tasks are counted by their first filing. (b) The last month of 2024 visible to the agent in the 99 tasks with a partial view. A forecast made from data up to a cutoff is graded against the months after it.

agent's answer or an intermediate result, whereas a filing states the value, interval, or reason the agent commits to, which also makes partial credit well defined. Fourth, grading is objective, unlike the LLM-as-a-judge evaluation (Zheng et al., 2023) used in many related works.

As a result, 178 of the 210 tasks act on accounts, file a forecast, allocate a budget, or publish a dashboard data source, and 99 see the warehouse only up to a cutoff month, as an analyst would at that date, so that forecasts are graded against months the agent has not seen (Figure 3). Each task is a workflow rather than a single query. Prompts average 159 words, 50 tasks require more than one filing, and the reference solution to the forecasting task in Figure 1 joins six tables across three EBS modules before it recovers the minimum-pay rule and fits an interval. In size, Argo-Bench sits with benchmarks of long-horizon agent work, such as TheAgentCompany (175 tasks) (Xu et al., 2025a), τ-bench (165) (Yao et al., 2025), KramaBench (104) (Lai et al., 2026), and ELT-Bench (100) (Jin et al., 2025).

Each expectation is scored from 0 to 100 by one of nine grading modes (Appendix G). Forecasts are scored by the skill of their weighted interval score over a reference forecast (Bracher et al., 2021), ban lists by the cost they save relative to banning nobody or everybody (Elkan, 2001), budget allocations by the share of the attainable savings that the simulator realizes, and data sources and reported figures by their values. A task's score is the mean of its expectations' scores, weighted as the task specifies. Optional expectations, such as a written note, are recorded but not scored, and a run that files nothing where the key expects action scores zero on every expectation.

We release one public seed (42) on Hugging Face and keep a private seed (43) for official grading. We ran our experiments on BigQuery, but the released warehouse is a set of 1,219 Parquet files partitioned by month (71.3 GiB compressed), and its dataset card gives setup instructions for DuckDB, Snowflake, Trino, Delta Lake, and Iceberg as well. The two seeds share the simulator and its calibration, but every ID, customer, restaurant, and courier differs.[5] The private seed keeps models from scoring by memorizing public answers. Table 1 compares Argo-Bench with prior benchmarks.

3 Evaluation

Experiments were executed on the open-source Inspect AI 0.3.263 harness (UK AI Security Institute, 2024) to prevent harness bias effects. The runs were executed on Google Kubernetes Engine (GKE), wherein agents were provided gVisor (Young et al., 2019) sandboxes with a Python environment, containing 25 preinstalled libraries, access to an empty file system, and no outbound internet access. Agents file to mission control through a Python library in their sandbox (`mission_control.py`), emulating a company's internal tooling, and every sandbox has its own mission control instance. A run ends after 500 model turns, which prevents "doom loops" wherein models repeat without making progress, and Appendix F lists the other limits and how runs

[5]Because the economy depends on the seed, volumes also differ slightly. The public warehouse has 7.54 billion rows against the 7.49 billion of the private world described in this paper.

Table 1: Argo-Bench puts an agent in an ERP-scale warehouse, lets it use Python and machine learning, and grades its actions against a latent state. Tables and rows are per database. For Spider 2.0, tables per database are for Spider 2.0-Lite as computed by Chen et al. (2024). Entries marked n/r are not reported by the benchmark. A coherent system is a single enterprise application whose tables must reconcile. Actions are graded when the benchmark scores what the agent changes or files rather than an answer it returns.

Benchmark	Tables per DB	Rows per DB	Data	Coherent system	Python and ML	Actions graded	Ground truth
BIRD (Li et al., 2023)	7.3	549K	Public	✗	✗	✗	Gold SQL
Spider 2.0 (Lei et al., 2025)	52.6	n/r	Public	✗	✗	✗	Gold SQL
BEAVER (Chen et al., 2024)	101.5	n/r	Private	✓	✗	✗	Logged SQL
DSBench (Jing et al., 2025)	n/r	n/r	Public	✗	✓	✗	Answer keys
DABstep (Egg et al., 2025)	n/r	138K	Real	✗	✓	✗	Answer keys
τ-bench (Yao et al., 2025)	3	2.8K	LLM-made	✗	✗	✓	Goal state
CRMArena-Pro (Huang et al., 2026)	25	55K	LLM-made	✓	✗	✗	Generator
Argo-Bench	**235**	**7.49B**	Simulated	✓	✓	✓	Latent state

Table 2: Results on Argo-Bench (210 tasks), each model at high reasoning effort where offered. Full is the share of tasks solved completely (a task score of at least 95), and score is the mean task score out of 100. Per-domain columns give the mean score in forecasting, fraud & risk, finance & accounting, dashboards, and compliance & analytics. Steps (model calls) and cost (model API spend, excluding BigQuery, which Appendix K reports) are averaged per task. Best results are in **bold**.

	Overall		Score by domain						
Model	Full (%)	Score	Fcst.	Fraud	Fin.	Dash.	Comp.	Steps	Cost ($)
Proprietary									
GPT-6 Astra	26.2	52.8	**48.5**	47.7	81.4	60.9	82.8	22	2.33
GPT-6 Sol	16.2	38.9	36.3	39.1	65.2	32.3	76.2	32	0.49
GPT-6 Luna	5.2	21.2	22.4	19.0	40.4	10.6	52.4	34	**0.04**
Claude Opus 5.5	**32.9**	**59.3**	45.6	**58.0**	**90.0**	**74.0**	**97.5**	54	2.33
Claude Sonnet 5	5.2	22.4	23.6	23.7	39.1	6.8	52.4	48	1.45
Claude Haiku 4.5	1.4	10.0	12.3	9.5	19.2	0.0	35.7	32	0.20
Gemini 3.8 Flash	10.0	16.3	14.4	5.1	56.4	10.8	71.4	140	2.45
Muse Spark 1.3	11.9	30.6	30.2	25.8	43.6	26.3	90.5	173	3.90
Open-weight									
Kimi K3	11.9	30.7	26.1	31.5	56.6	24.2	71.4	53	1.62
GLM 5.3 Flash	8.1	25.8	26.4	23.6	42.9	17.3	61.9	68	0.16
DeepSeek V4.1 Flash	9.0	30.0	27.1	32.3	40.4	18.7	82.1	67	0.13
Qwen 3.8 Max	17.6	34.3	32.9	23.3	66.1	35.4	65.5	77	2.69

that reach them are scored. We selected latest-generation models as of September 2026 across price ranges (Table 2).

3.1 Results

For OpenAI, Anthropic, Google, and Meta models, the respective APIs were used directly, and all others were accessed via Fireworks serverless endpoints with default settings. Claude Opus 5.5 leads overall and in four of the five domains, and GPT-6 Astra leads forecasting (Table 2). More reasoning effort helps the four models we follow across low, medium, and high effort, with diminishing returns: Opus gains 14.7 points from low to medium effort but only 3.3 from medium to high. It does not help every model, and Gemini 3.8 Flash scores lower at high effort than at low (Appendix I). Gemini 3.8 Flash and Muse Spark 1.3 average 140 and 173 model calls per task against 54 for Opus without scoring higher, and their warehouse queries follow the same pattern: Muse Spark scans 1.7 times as many bytes per task as the next model, and 76% of Gemini's BigQuery spend goes to tasks that score below 5 out of 100 (Appendix K). Bootstrap 95% confidence intervals over tasks are about ± 5

points on the score and ±5 to 7 points on the solved rate, so Opus and Astra separate clearly from the other models while the intervals of the next five overlap (Appendix I).

3.2 FINDINGS

The traces show agents running sophisticated analyses and then measuring the wrong quantity, optimizing the wrong outcome, or learning from the wrong labels (Appendix I).

Right analysis, wrong objective. The quest-cut tasks ask where to remove $9.6 million from the annual bonus bill for couriers. Without hints, GPT-6 Astra (xhigh) finds the eligible cells that were never launched, estimates supply responses with fixed effects and partial pooling, and files a plan within budget that cuts where quests buy the fewest courier-hours. The platform pays for quests to avoid surge pay, however, so under the simulator's response model the plan loses $86,281 where a uniform cut would save $0.40 million, and it scores 0. Claude Opus 5.5 (xhigh) finds that quests substitute for surge pay, cuts where they save the least surge per bonus dollar, saves $3.09 million of an attainable $3.12 million, and scores 99. A grader that checked only the budget would pass both plans. Stating the purpose of quests and the existence of the holdout raises the mean score over 30 settings from 16.7 to 59.9.

Right data, wrong quantity. Many failures measure a quantity other than the one asked for. All 30 completed runs of a task that sizes the courier location service miss all 12 monthly counts despite a 1% tolerance, because the warehouse keeps only hourly idle check-ins while the app sends them every half hour. GPT-6 Astra (xhigh) notices the gap, writes that its count does not establish how many reports the app sent, and files it anyway. On orders turned away by full kitchens, no run passes (0 of 30) until the prompt states that the event is the platform's refusal rather than a merchant's rejection (11 of 30). Across dashboard tasks, 72.9% of the data sources that pass their structural contract score zero on their values.

Past labels and benign lookalikes in fraud. To find couriers who steal orders, GPT-6 Astra trains four classifiers on past enforcement and reaches a temporal-holdout AUC of .932, then acts on one courier and catches none of the 78 targets. Past deactivations record the platform's review process, not verified labels. On a shorter prompt, the same setting inspects delivery evidence directly and catches 40 of the 78 without an innocent ban. In the refund-collusion family, mean scores over 30 settings fall from 32.6 to 9.8 as the search widens from mid-sized storefronts to every storefront and the rings grow from eight to ten. On the full search, Claude Opus 5.5 (xhigh) catches all ten rings but holds 74 innocent storefronts, because it reads a benign pattern of early claims by regular customers as a second kind of ring.

Overconfident forecasts. Across 2,789 forecast series from 2,055 runs on 72 tasks, nominal 80% intervals contain the realized value only 42.2% of the time. Skill scores also depend on the reference. Rescored against the tighter reference of their harder variant, with points, intervals, and outcomes unchanged, the June base-pay forecasts fall from a mean score of 74.6 to 0.9, although their median absolute error is 1.86%. Appendix I therefore reports coverage and error alongside skill.

4 RELATED WORK

Text-to-SQL and data science benchmarks. Text-to-SQL benchmarks have moved from databases with a handful of tables each (Yu et al., 2018; Li et al., 2023) to enterprise-scale schemas (Lei et al., 2025) and private data warehouses (Chen et al., 2024), and data science benchmarks extend evaluation to multi-step analysis, data lakes, and data pipelines (Jing et al., 2025; Egg et al., 2025; Lai et al., 2026; Lei et al., 2026; Jin et al., 2025). These benchmarks compare an agent's output to a gold answer, which audits have found to be frequently wrong (Wretblad et al., 2024; Jin et al., 2026), or to expert conclusions, often scored by an LLM judge (Gu et al., 2024; Sahu et al., 2025). Argo-Bench instead grades against the simulator's latent state, and it grades actions rather than answers.

Agent benchmarks in simulated environments. Simulated environments are standard for evaluating agents that act. AppWorld (Trivedi et al., 2024) and τ-bench (Yao et al., 2025; Barres et al.,

2025) check the final state of the environment's database, TheAgentCompany (Xu et al., 2025a) scores checkpoints in a simulated software company, and CRMArena-Pro (Huang et al., 2026) shapes LLM-generated CRM records with latent variables. Vending-Bench and Business Arena score the net worth of a business that the agent runs (Backlund & Petersson, 2025; Pan et al., 2026). In these benchmarks, the state that determines success is either observable to the agent or changed by the agent inside a stylized game. In Argo-Bench, it is withheld and must be reconstructed from an enterprise warehouse. Simulators calibrated to public statistics have likewise supplied ground truth that real data lacks for patient records (Walonoski et al., 2018) and money laundering (Altman et al., 2023).

Generated enterprises with hidden ground truth. AvalancheBench (Kłeczek et al., 2026) uses an LLM judge to score how much of a small latent e-commerce world an agent's report recovers. The Era by Eon benchmark (Gruenbaum et al., 2026) serves a generated company through a fleet of 66 simulated products, calibrated to operational and published statistics, and plants the records that answer each of its read-only questions, together with near misses. Two unrelated benchmarks named ERPBench evaluate decisions in a simulated manufacturer (Zhang et al., 2026) and computer-use tasks in a live ERP system (Bhagtani et al., 2026). In contrast, the latent state of Argo-Bench is produced by the simulation itself rather than planted, its evidence is spread across an enterprise warehouse of 7.49 billion rows, and agents are graded on the consequences of the actions they file rather than on the answers they give.

5 Limitations & Future Work

The dataset still differs from the most complex ERP deployments in four ways. First, foreign currency and its representations, namely transaction, local, and group or reporting currency, are not represented, since the dataset covers a single city. Second, only one business line is represented, while many complex warehouses combine several, such as food delivery, grocery delivery, and ride sharing, in the same balance sheet accounts. Third, only one year is simulated. A longer history, such as 2017 to 2026, would add the market shocks of 2020 to 2022 to operations and forecasts. Fourth, only one ERP format is supported, and a natural extension of this benchmark would add an SAP S/4HANA flavor alongside Oracle EBS.

The simulator validates 23 distinct metrics, and we keep behavior that public figures do not constrain out of scope for tasks. The in-world membership program in particular rests on weakly grounded assumptions. More generally, a simulator encodes the assumptions of its authors, and a generator that shares the simplifying assumptions of the systems under evaluation can make tasks easier than their real counterparts (Leis et al., 2015). Calibration to aggregate targets also does not guarantee realistic tails (Chen et al., 2019). We therefore present Argo-Bench as a measure of the relative capabilities of data agents rather than of their real-world performance.

Each model has one run per task, and all tasks share one simulated world, so our confidence intervals reflect the choice of tasks rather than run-to-run variation. Plan tasks are graded under the simulator's frozen response model, and tasks in which the agent changes the live platform, such as repricing deliveries, await re-simulation in a future release. Appendix J lists the open issues in individual tasks.

6 Conclusion

We introduced Argo-Bench, a benchmark that evaluates data agents on a simulated food delivery platform projected into an Oracle E-Business Suite warehouse of 235 tables and 7.49 billion rows. By withholding the simulator's latent state from the warehouse, Argo-Bench grades agents on the facts they reconstruct and the actions they file rather than on the SQL they write, and it computes every answer key from the ground truth rather than from annotators. The strongest of 12 models solves 32.9% of tasks and averages 59.3 points. The most instructive failures are analyses of the wrong quantity, toward the wrong objective, or trained on the wrong historical labels. We release the public seed's warehouse, tasks, reference solutions, and harness, and hope that Argo-Bench helps measure progress towards data agents that can understand, navigate, and act within real enterprise data environments.

AI USE STATEMENT

Large language models were used in three ways. First, we used them in building the simulator, to iterate on parameter tuning, to clean donor datasets, and to reconcile restaurant records where Overture and the DOHMH inspections disagree (Section 2.1). No record in the simulated world or the warehouse is generated by a language model, and every value comes from the simulator. Second, we used them as coding assistants for developing the simulator, graders, task prompts, reference solutions, and figure scripts. The task prompts are either human-written or AI-generated, curated and re-written by a human. Third, we used them to edit the text of the paper, check its citations, and draft figure and table code. The authors checked every claim, number, and citation. The authors take full responsibility for the content of the paper.

ETHICS STATEMENT

Argo-Bench contains no data about real people. Consumer and courier names are drawn from public name-frequency tables and assigned to simulated people, and every order, shift, and payment is simulated. Restaurants are real New York City businesses taken from public records, with their names and addresses. All behavior attributed to them in the world, including the fraud patterns of Section 2.1, is simulated and says nothing about the real businesses. The donor datasets are used under their licenses. Those released for research or non-commercial use, such as the Yelp Open Dataset and the Grubhub MDRP instances, serve only as shape donors, and none of their records enters the world or the warehouse (Appendix A). The fraud tasks reward detecting common schemes, not carrying them out. During development, one model escaped an insufficiently isolated sandbox and read the grader code (Appendix D). The final runs use isolated sandboxes without internet access, and we report the incident so that others building agent benchmarks can guard against it.

REPRODUCIBILITY STATEMENT

Two artifacts are released. The warehouse of one seed is released on Hugging Face under CC BY 4.0 at `https://huggingface.co/datasets/DoubleBlindAnon912/iclr-double-blind-dataset`. The code that reproduces the paper's runs is released under the Apache License 2.0 at `https://anonymous.4open.science/r/Argo-Bench-8B75` (anonymized for review). It contains the 210 questions, the reference agent on Inspect AI 0.3.263, its warehouse and Python tool servers, the mission-control console the agent files through, the three sandboxes `run_python` executes in (a macOS Seatbelt profile, a Docker image with pinned libraries, and the network-less gVisor pod used for the paper's runs), loaders for DuckDB and BigQuery, and the model and reasoning-effort configuration of every rung reported. The system prompt and tools are given in Appendices E and F, and Appendix L shows three reference solutions in full.

The simulator, the answer keys, and the graders are not released, so the benchmark cannot be trained on. A run made with the released code exports a submission file recording every filing, the model and its settings, token usage, and how each run ended, which we score on a best-effort basis. The paper's runs queried a second world generated by the same simulator from a different seed (Section 2.3). The 15 questions that name specific couriers, storefronts, or promotion codes draw them from the public world by the same selection rule, and their cards say so. Re-running the released code therefore reproduces the paper's procedure on a sibling world rather than its exact numbers. Proprietary models were accessed through their providers' APIs and all other models through Fireworks serverless endpoints with default settings, in September 2026, and results from proprietary APIs may drift as providers update their models.

REFERENCES

Al Jazeera. Delivery driver pleads guilty to stealing $2.5m from DoorDash. `https://www.aljazeera.com/economy/2025/5/14/delivery-driver-pleads-guilty-to-stealing-2-5m-from-doordash`, 2025. 2025-05-14.

Erik Altman, Jovan Blanuša, Luc von Niederhäusern, Béni Egressy, Andreea Anghel, and Kubilay Atasu. Realistic synthetic financial transactions for anti-money laundering models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 29851–29874, 2023.

Anthropic. Introducing the Model Context Protocol. https://www.anthropic.com/news/model-context-protocol, 2024.

Axel Backlund and Lukas Petersson. Vending-Bench: A benchmark for long-term coherence of autonomous agents. *arXiv preprint arXiv:2502.15840*, 2025.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Kratika Bhagtani, Kusha Sridhar, Maziyar Baran Pouyan, Yuying Zhao, and Eugene Siow. ERP-Bench: A state-grounded evaluation paradigm for computer-use agents in enterprise software. *arXiv preprint arXiv:2609.17885*, 2026.

Johannes Bracher, Evan L Ray, Tilmann Gneiting, and Nicholas G Reich. Evaluating epidemic forecasts in an interval format. *PLoS Computational Biology*, 17(2):e1008618, 2021.

Lars Brehm, Armin Heinzl, and M Lynne Markus. Tailoring ERP systems: a spectrum of choices and their implications. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*. IEEE, 2001.

Lizette Chapman and Kartikay Mehrotra. Instacart shoppers say they are battling order grabbing bots that cut their profits. https://fortune.com/2020/08/01/instacart-shoppers-bots-orders, 2020. Bloomberg via Fortune, 2020-08-01.

Junqiao Chen, David Chun, Milesh Patel, Epson Chiang, and Jesse James. The validity of synthetic clinical data: a validation study of a leading synthetic data generator (Synthea) using clinical quality measures. *BMC Medical Informatics and Decision Making*, 19(1):44, 2019.

Peter Baile Chen, Devin Yang, Weiyue Li, Fabian Wenz, Yi Zhang, Nesime Tatbul, Michael Cafarella, Çağatay Demiralp, and Michael Stonebraker. BEAVER: An enterprise benchmark for Text-to-SQL. *arXiv preprint arXiv:2409.02038*, 2024.

Yeounoh Chung, Gaurav T. Kakkar, Yu Gan, Brenton Milne, and Fatma Özcan. Is long context all you need? Leveraging LLM's extended context for NL2SQL. *Proceedings of the VLDB Endowment*, 18(8):2735–2747, 2025.

Thomas H Davenport. Putting the enterprise into the enterprise system. *Harvard Business Review*, 76(4):121–131, 1998.

DoorDash, Inc. Annual report (Form 10-K) for the fiscal year ended December 31, 2022. https://www.sec.gov/Archives/edgar/data/1792789/000162828023005131/dash-20221231.htm, 2023. Filed 2023-02.

DoorDash, Inc. DoorDash further strengthens safeguards against account sharing. https://about.doordash.com/en-us/news/doordash-further-strengthens-safeguards-against-account-sharing, 2024a. 2024-12-12.

DoorDash, Inc. Annual report (Form 10-K) for the fiscal year ended December 31, 2023. https://www.sec.gov/Archives/edgar/data/1792789/000162828024005600/0001628280-24-005600-index.htm, 2024b. Filed 2024-02-20.

DoorDash, Inc. Annual report (Form 10-K) for the fiscal year ended December 31, 2024. https://www.sec.gov/Archives/edgar/data/1792789/000162828025005715/0001628280-25-005715-index.htm, 2025. Filed 2025-02-14.

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

Charles Elkan. The foundations of cost-sensitive learning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 973–978. Morgan Kaufmann, 2001.

Andrea Gadotti, Luc Rocher, Florimond Houssiau, Ana-Maria Creţu, and Yves-Alexandre de Montjoye. Anonymization: The imperfect science of using data while preserving privacy. *Science Advances*, 10(29):eadn7053, 2024. doi: 10.1126/sciadv.adn7053.

Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. BigBench: Towards an industry standard benchmark for big data analytics. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pp. 1197–1208, 2013.

Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

Grubhub Inc. Annual report (Form 10-K) for the fiscal year ended December 31, 2020. `https://www.sec.gov/Archives/edgar/data/1594109/000156459021009522/0001564590-21-009522-index.htm`, 2021. Filed 2021-03-01.

Benjamin Gruenbaum, Doron Porat, Assaf Natanzon, Roy Zavida, Chen Dinachi, and Or Itzahary. The Era by Eon benchmark: A generated enterprise estate with exact ground truth for benchmarking LLM agents. *arXiv preprint arXiv:2609.09853*, 2026.

Ken Gu, Ruoxi Shang, Ruien Jiang, Keying Kuang, Richard-John Lin, Donghe Lyu, Yue Mao, Youran Pan, Teng Wu, Jiaqian Yu, Yikun Zhang, Tianmai M. Zhang, Lanyi Zhu, Mike A Merrill, Jeffrey Heer, and Tim Althoff. BLADE: Benchmarking language model agents for data-driven science. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 13936–13971, 2024.

Bernadette Heier. Uber Eats to remove thousands of duplicate virtual brands. `https://foodondemand.com/03302023/uber-eats-to-remove-thousands-of-duplicate-virtual-brands/`, 2023. Food On Demand, 2023-03-30.

Kung-Hsiang Huang, Akshara Prabhakar, Onkar Thorat, Divyansh Agarwal, Prafulla Kumar Choubey, Yixin Mao, Silvio Savarese, Caiming Xiong, and Chien-Sheng Wu. CRMArena-Pro: Holistic assessment of LLM agents across diverse business scenarios and interactions. *Transactions on Machine Learning Research*, 2026.

Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 3rd edition, 2021.

Incognia. Incognia mobile app fraud insights report reveals food delivery apps are major target for location-based fraud. `https://www.incognia.com/newsroom/incognia-mobile-app-fraud-insights-report-reveals-food-delivery-apps-are-major-target-for-location-based-fraud`, 2022. 2022-08-30.

Incognia. Incognia's gig economy fraud report shows refund abuse representing 48% of consumer fraud in 2024. `https://www.incognia.com/newsroom/gig-economy-fraud-report-shows-refund-abuse-representing-48-percent-of-consumer-fraud-in-2024`, 2025. 2025-02-26.

Tengjun Jin, Yuxuan Zhu, and Daniel Kang. ELT-Bench: An end-to-end benchmark for evaluating AI agents on ELT pipelines. *Proceedings of the VLDB Endowment*, 19(2):84–98, 2025.

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.

Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. DSBench: How far are data science agents from becoming data science experts? In *International Conference on Learning Representations*, volume 2025, pp. 32597–32649, 2025.

Sean Kandel, Andreas Paepcke, Joseph M Hellerstein, and Jeffrey Heer. Enterprise data analysis and visualization: An interview study. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2917–2926, 2012.

Darek Kłeczek, Fuheng Zhao, Alexander W. Lee, Julien Tissier, Paweł Liskowski, Uğur Çetintemel, and Anupam Datta. AvalancheBench: Evaluating enterprise data agents through latent world recovery. *arXiv preprint arXiv:2605.24183*, 2026.

Eugenie Lai, Gerardo Vitagliano, Ziyu Zhang, Om Chabra, Sivaprasad Sudhir, Anna Zeng, Anton Zabreyko, Chenning Li, Ferdi Kossmann, Jialin Ding, Jun Chen, Markos Markakis, Matthew Russo, Weiyang Wang, Ziniu Wu, Mike Cafarella, Lei Cao, Samuel Madden, and Tim Kraska. KramaBench: A benchmark for AI systems on data-to-insight pipelines over data lakes. In *International Conference on Learning Representations*, volume 2026, pp. 142883–142912, 2026.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating language models on real-world enterprise Text-to-SQL workflows. In *International Conference on Learning Representations*, volume 2025, pp. 28691–28735, 2025.

Fangyu Lei, Jinxiang Meng, Yiming Huang, Junjie Zhao, Yitong Zhang, Jianwen Luo, Xin Zou, Ruiyi Yang, Wenbo Shi, Yan Gao, Shizhu He, Jun Zhao, Zuo Wang, Qian Liu, Yang Wang, Ke Wang, and Kang Liu. DAComp: Benchmarking data agents across the full data intelligence lifecycle. In *International Conference on Learning Representations*, volume 2026, pp. 104463–104501, 2026.

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.

Boyan Li, Yiran Peng, Yupeng Xie, Sirong Lu, Yizhang Zhu, Xing Mu, Xinyu Liu, and Yuyu Luo. DeepEye: A steerable self-driving data agent system. In *Companion of the International Conference on Management of Data*, pp. 74–77, 2026.

Jiacheng Li, Jingbo Shang, and Julian McAuley. UCTopic: Unsupervised contrastive learning for phrase representations and topic mining. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6159–6169, 2022.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as a database interface? A BIg bench for large-scale database grounded Text-to-SQLs. In *Advances in Neural Information Processing Systems*, volume 36, pp. 42330–42357, 2023.

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 *Conference on Innovative Data Systems Research (CIDR)*, 2026.

Edgar Lopez-Rojas, Ahmad Elmir, and Stefan Axelsson. PaySim: A financial mobile money simulator for fraud detection. In *28th European Modeling and Simulation Symposium (EMSS)*, pp. 249–255, 2016.

Khalil Maycock. Jacksonville restaurant loses thousands after DoorDash account hacked. `https://www.news4jax.com/news/local/2024/12/02/small-business-loses-thousands-after-doordash-account-hacked/`, 2024. News4JAX, 2024-12-02.

Allan H Murphy. Skill scores based on the mean square error and their relationships to the correlation coefficient. *Monthly Weather Review*, 116(12):2417–2424, 1988.

Raghunath Othayoth Nambiar and Meikel Poess. The making of TPC-DS. In *International Conference on Very Large Data Bases (VLDB)*, pp. 1049–1058, 2006.

NYC Department of Consumer and Worker Protection (DCWP). Restaurant delivery app data: Quarterly reports, Q1–Q4 2024. `https://www.nyc.gov/site/dca/workers/Delivery-Worker-Public-Hearing-Minimum-Pay-Rate.page`, 2024a. Accessed 2026-09.

NYC Department of Consumer and Worker Protection (DCWP). Mayor Adams announces first annual increase in minimum pay rate for app-based restaurant delivery workers. https://www.nyc.gov/site/dca/news/018-24/mayor-adams-first-annual-increase-minimum-pay-rate-app-based-restaurant-delivery, 2024b.

Yijun Pan, Yukun Lian, Kunyu Shi, Junbo Li, Hongwei Xue, Sicong Xie, Guannan Zhang, and Xiaoying Xing. Business Arena: Benchmarking LLM agents in a realistic marketplace. *arXiv preprint arXiv:2608.08621*, 2026.

Hasso Plattner. The impact of columnar in-memory databases on enterprise systems: implications of eliminating transaction-maintained aggregates. *Proceedings of the VLDB Endowment*, 7(13): 1722–1729, 2014.

Gaurav Sahu, Abhay Puri, Juan A. Rodriguez, Amirhossein Abaskohi, Mohammad Chegini, Alexandre Drouin, Perouz Taslakian, Valentina Zantedeschi, Alexandre Lacoste, David Vazquez, Nicolas Chapados, Christopher Pal, Sai Rajeswar, and Issam Laradji. InsightBench: Evaluating business analytics agents through multi-step insight generation. In *International Conference on Learning Representations*, volume 2025, pp. 4683–4715, 2025.

Joar Skalse, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. In *Advances in Neural Information Processing Systems*, volume 35, pp. 9460–9471, 2022.

Yuda Song, Hanlin Zhang, Carson Eisenach, Sham Kakade, Dean Foster, and Udaya Ghai. Mind the gap: Examining the self-improvement capabilities of large language models. In *International Conference on Learning Representations*, volume 2025, pp. 39894–39931, 2025.

Zhaoyan Sun, Jiayi Wang, Xinyang Zhao, Jiachi Wang, and Guoliang Li. Data agent: A holistic architecture for orchestrating data+AI ecosystems. *arXiv preprint arXiv:2507.01599*, 2025.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

UK AI Security Institute. Inspect AI: Framework for large language model evaluations. https://github.com/UKGovernmentBEIS/inspect_ai, 2024.

Alexander van Renen, Dominik Horn, Pascal Pfeil, Kapil Vaidya, Wenjian Dong, Murali Narayanaswamy, Zhengchun Liu, Gaurav Saxena, Andreas Kipf, and Tim Kraska. Why TPC is not enough: An analysis of the Amazon Redshift fleet. *Proceedings of the VLDB Endowment*, 17(11):3694–3706, 2024.

Adrian Vogelsgesang, Michael Haubenschild, Jan Finis, Alfons Kemper, Viktor Leis, Tobias Mühlbauer, Thomas Neumann, and Manuel Then. Get real: How benchmarks fail to represent the real world. In *Proceedings of the Workshop on Testing Database Systems*, pp. 1–6, 2018.

Jason Walonoski, Mark Kramer, Joseph Nichols, Andre Quina, Chris Moesel, Dylan Hall, Carlton Duffett, Kudakwashe Dube, Thomas Gallagher, and Scott McLachlan. Synthea: An approach, method, and software mechanism for generating synthetic patients and the synthetic electronic health care record. *Journal of the American Medical Informatics Association*, 25(3):230–238, 2018.

Zack Whittaker. DoorDash customers say their accounts have been hacked. https://techcrunch.com/2018/09/25/doordash-customers-say-their-accounts-have-been-hacked, 2018. TechCrunch, 2018-09-25.

Niklas Wretblad, Fredrik Riseby, Rahul Biswas, Amin Ahmadi, and Oskar Holmström. Understanding the effects of noise in Text-to-SQL: An examination of the BIRD-Bench benchmark. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 356–369, 2024.

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. TheAgentCompany: Benchmarking LLM agents on consequential real world tasks. In *Advances in Neural Information Processing Systems*, volume 38, 2025a.

Muxi Xu, Kun Hu, Sudeep Das, and Bruce Wang. Causal machine learning for promotions: Industry evidence and applications. In *KDD Workshop on Causal Inference and Machine Learning in Practice*, 2025b.

An Yan, Zhankui He, Jiacheng Li, Tianyang Zhang, and Julian McAuley. Personalized showcases: Generating multi-modal explanations for recommendations. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2251–2255, 2023.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*, volume 2025, pp. 9965–10017, 2025.

Yelp Inc. Yelp Open Dataset. `https://business.yelp.com/data/resources/open-dataset/`, 2026. Accessed 2026-09.

Ethan G Young, Pengfei Zhu, Tyler Caraza-Harter, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. The true cost of containing: A gVisor case study. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, 2019.

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*, pp. 3911–3921, 2018.

Xinran Zhang, Pengrui Lu, Lyumanshan Ye, and Pengfei Liu. ERPBench: Evaluating LLM agents for enterprise decision-making across competitive market ecologies. *arXiv preprint arXiv:2609.04667*, 2026.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623, 2023.

A DONOR DATASETS

L Table 3 lists the roughly fifty public datasets and reports the world is built from (Section 2.1), by the part of the world they build. Donors released for research or non-commercial use, among them the Yelp Open Dataset (Yelp Inc., 2026) and the Google Local reviews (Li et al., 2022; Yan et al., 2023), serve only as shape donors: none of their records reaches the world or the released warehouse, only parameters fitted on them.

B WAREHOUSE TABLES

Table 4 lists every table in the warehouse with its column and row counts, grouped by the Oracle E-Business Suite module that owns it. The 159 standard tables fall under 15 Oracle products. The 76 custom extensions are grouped by the part of the business they record, because their `XX_` prefix does not say.

C QUESTIONS ENABLED BY THE MINIMUM-PAY CHANGE

On April 1, 2024, NYC raised the minimum pay rate for app-based restaurant delivery workers from $17.96 to $19.56 per hour (DCWP, 2024b), and the simulated platform responds on the same day. Its dispatcher starts restricting when couriers can go online, and its per-delivery pay card drops (Section 2.1). Whenever a week's pay falls short of the minimum, the platform tops it up with a true-up on the weekly courier pay invoice. The shock breaks the stationarity that simple forecasts rely on, and it changes how several standard figures must be computed. Table 5 groups the tasks that depend on it.

D REWARD HACKING

During earlier experimental runs, the Python code execution environments were not sufficiently isolated, and we detected at least three successful reward hacking (Skalse et al., 2022) attempts from Gemini 3.8 Flash, wherein the model successfully changed its root, located the grader code for the question under assessment (`fraud-04`), and managed to score a 0.99. It is notable that every other model under evaluation had access to this exploit but did not utilize it. For the final runs, we adapted our environments to use Inspect's Kubernetes sandbox provider in a Google Kubernetes Engine environment with no outbound internet access, as well as to provide restrictive BigQuery service accounts that only allow queries to the dataset selected for the question (the full-year view or a monthly view).

E SYSTEM PROMPT

```
You are an analyst for a food delivery platform operating in NYC. You
    have read-only access to a transformed export of the company's
    Oracle E-Business Suite warehouse.
You should explore the warehouse, investigate, and then perform
    appropriate actions based on your findings.

Warehouse

One Oracle EBS instance, order-to-cash, covering the calendar year
    2024, following Oracle house conventions.
Timestamps are naive America/New_York: compare "as of 2024-07-31
    23:59:59 ET" directly against stored values, without converting.

Mission Control

File every finding from Python (`run_python`) through the Mission
    Control console. A number stated only in prose is not filed.
Example:
```

Table 3: The donor datasets, by the part of the world they build. Role: I = identity (real NYC records, used as they are), S = shape (a distribution measured elsewhere and transferred), A = anchor (published figures the world is calibrated to). † Released for research or non-commercial use, and used only as a shape donor. ‡ Used for validation only: no fit reads it.

Donor	What it lends the world	Role
The city		
Overture Places	The 45,834 restaurants, with names and locations	I
DOHMH inspections	Cuisine, license, grades, openings and closures since 2017	I
Overture Addresses	1.07M residential address points	I
MapPLUTO	Building use, units, and floor area of 857k tax lots	I
TLC taxi zones	The 260 zones the world is organized by	I
OpenStreetMap	Street network and bridge access	I
ACS 2020–2024	Tract composition, households, and zone income	I
The customers		
2020 Census surnames	Surnames by origin	S
WGND 2.0	Given names by country and gender	S
NYC baby names	Given names by NYC community	S
SSA New York births	Given names of the cohorts adult in 2024	S
NANPA code assignments	Phone exchanges assigned in the NYC area	I
LODES 2023	Home-to-work flows for office anchors	S
When they order		
TLC high-volume FHV trips	Demand by zone, hour, and weekday	S
GHCN-Daily Central Park	Daily rain, snow, and temperature	I
NOAA LCD	Hourly precipitation	I
DoorDash 2015 export	Event-day demand shapes	S
ATUS 2024	How often and when people host guests	S
What they order and pay		
Google Local reviews†	Storefront popularity and quality rank	S
Yelp Open Dataset†	Popularity curve and check-in tempo by cuisine	S
MenuStat	Chain menus	I
Wayback Machine offers	1,648 promotions: values, limits, durations	S
The kitchen		
Grubhub MDRP†	Preparation times at named restaurants	S
QSR drive-thru study	The fast end of preparation times	A
NYC Open Restaurants	Seating, which bounds kitchen load	I
The courier		
Chicago taxi trips	Shift lengths and trips per shift	S
NYC Citi Bike	E-bike speeds by street class	S
NYC yellow cabs, 2013–2015	Where within a zone a courier stops	S
DCWP worker study 2022	Vehicle mix and connected time	A
Fraud and the books		
IEEE-CIS†	Fraud separability and reuse; email domains	S
RBA logins	How often honest users share devices and networks	S
UCI Online Retail II	Returns and partial refunds	S
DCWP quarterly reports	Orders, fees, and courier pay	A
SEC 10-K filings	Balance-sheet and expense shapes	A

Table 4: The warehouse's 235 tables, 3,949 columns and 7,491,508,904 rows, by Oracle E-Business Suite module. Families and colours follow Figure 2b. Standard tables sit under the Oracle product that owns them; custom XX_ extensions are grouped by what they record. Bars show rows on a log scale. The six empty tables are four interface tables, which a completed import leaves empty, AR_ADJUSTMENTS_ALL, and HZ_CUST_ACCOUNT_ROLES.

Orders & parties — 51 tables · 723 M rows

Table	Cols	Rows
Order Management	6 tables	580 M rows
OE_ORDER_HEADERS_ALL	41	81 M
OE_ORDER_LINES_ALL	24	446 M
OE_ORDER_SOURCES	12	40
OE_PRICE_ADJUSTMENTS	35	52.9 M
OE_TRANSACTION_TYPES_ALL	13	4
OE_TRANSACTION_TYPES_TL	13	4
Advanced Pricing	5 tables	135 k rows
QP_LIST_HEADERS_B	29	33.6 k
QP_LIST_HEADERS_TL	9	33.6 k
QP_LIST_LINES	33	33.8 k
QP_PRICING_ATTRIBUTES	23	33.9 k
QP_QUALIFIERS	24	2
Trading Community	12 tables	97.5 M rows
HZ_CODE_ASSIGNMENTS	18	28
HZ_CONTACT_POINTS	18	8.45 M
HZ_CUSTOMER_PROFILES	22	4.21 M
HZ_CUST_ACCOUNTS	14	4.21 M
HZ_CUST_ACCOUNT_ROLES	15	0
HZ_CUST_ACCT_SITES_ALL	17	18.5 M
HZ_CUST_PROFILE_CLASSES	21	1
HZ_CUST_SITE_USES_ALL	17	22.7 M
HZ_LOCATIONS	34	16.6 M
HZ_ORGANIZATION_PROFILES	17	17.9 k
HZ_PARTIES	31	4.31 M
HZ_PARTY_SITES	15	18.5 M
Contracts Core	9 tables	17.2 M rows
OKC_K_HEADERS_ALL_B	53	1.61 M
OKC_K_HEADERS_TL	11	1.61 M
OKC_K_ITEMS	21	2.51 M
OKC_K_LINES_B	49	2.51 M
OKC_K_LINES_TL	11	2.51 M
OKC_K_PARTY_ROLES_B	17	3.22 M
OKC_K_PARTY_ROLES_TL	9	3.22 M
OKC_STATUSES_B	11	8
OKC_STATUSES_TL	11	8
Service Contracts: subscriptions	6 tables	25.7 M rows
OKS_BILL_CONT_LINES	33	5.92 M
OKS_BILL_TXN_LINES	18	5.92 M
OKS_K_LINES_B	29	2.51 M
OKS_K_LINES_TL	10	2.51 M
OKS_LEVEL_ELEMENTS	19	6.31 M
OKS_STREAM_LEVELS_B	24	2.51 M
Inventory: items & catalog	9 tables	3.11 M rows
MTL_CATEGORIES_B	10	70
MTL_CATEGORIES_TL	9	70
MTL_CATEGORY_SETS_B	11	1
MTL_CATEGORY_SETS_TL	9	1
MTL_ITEM_CATEGORIES	13	1.03 M
MTL_PARAMETERS	12	17.9 k
MTL_SYSTEM_ITEMS_B	32	1.03 M
MTL_SYSTEM_ITEMS_TL	10	1.03 M
MTL_UNITS_OF_MEASURE_TL	17	1
Human Resources: organizations & sites	4 tables	53.6 k rows
HR_ALL_ORGANIZATION_UNITS	16	17.9 k
HR_LOCATIONS_ALL	17	17.9 k
HR_OPERATING_UNITS	6	1
HR_ORGANIZATION_INFORMATION	14	17.9 k

Finance — 79 tables · 4.18 B rows

Table	Cols	Rows
Receivables	22 tables	1.64 B rows
AR_ADJUSTMENTS_ALL	25	0
AR_AGING_BUCKETS	9	1
AR_AGING_BUCKET_LINES_B	11	5
AR_AGING_BUCKET_LINES_TL	9	5
AR_BATCHES_ALL	22	493
AR_CASH_RECEIPTS_ALL	39	85.8 M
AR_CASH_RECEIPT_HISTORY_ALL	25	258 M
AR_DISTRIBUTIONS_ALL	12	159 M
AR_PAYMENT_SCHEDULES_ALL	29	174 M
AR_RECEIPT_CLASSES	12	1
AR_RECEIPT_METHODS	9	2
AR_RECEIVABLES_TRX_ALL	12	2
AR_RECEIVABLE_APPLICATIONS_ALL	27	90.9 M
RA_BATCH_SOURCES_ALL	10	3
RA_CUSTOMER_TRX_ALL	26	88.9 M
RA_CUSTOMER_TRX_LINES_ALL	20	533 M
RA_CUST_TRX_LINE_GL_DIST_ALL	20	246 M
RA_CUST_TRX_TYPES_ALL	20	4
RA_INTERFACE_LINES_ALL	18	0
RA_TERMS_B	12	3
RA_TERMS_LINES	9	3
RA_TERMS_TL	10	3
Payables	12 tables	48.6 M rows
AP_CHECKS_ALL	22	5.25 M
AP_HOLDS_ALL	14	31.2 k
AP_HOLD_CODES	11	13
AP_INVOICES_ALL	36	6.06 M
AP_INVOICE_DISTRIBUTIONS_ALL	19	7.89 M
AP_INVOICE_LINES_ALL	17	17.4 M
AP_INVOICE_PAYMENTS_ALL	15	5.65 M
AP_PAYMENT_SCHEDULES_ALL	13	6.06 M
AP_SUPPLIERS	33	98.7 k
AP_SUPPLIER_SITES_ALL	19	98.7 k
AP_TERMS_LINES	9	3
AP_TERMS_TL	11	3
Payments	6 tables	11.2 M rows
IBY_DOCS_PAYABLE_ALL	31	5.65 M
IBY_EXTERNAL_PAYEES_ALL	12	98.7 k
IBY_EXT_BANK_ACCOUNTS	18	98.1 k
IBY_PAYMENTS_ALL	34	5.25 M
IBY_PAY_INSTRUCTIONS_ALL	23	1.07 k
IBY_PMT_INSTR_USES_ALL	14	101 k
Cash Management	10 tables	96.1 M rows
CE_BANK_ACCOUNTS	34	1
CE_BANK_ACCT_USES_ALL	21	1
CE_GL_ACCOUNTS_CCID	13	1
CE_STATEMENT_HEADERS	20	251
CE_STATEMENT_HEADERS_INT	13	0
CE_STATEMENT_LINES	19	5.4 M
CE_STATEMENT_LINES_INTERFACE	14	0
CE_STATEMENT_RECONCILS_ALL	18	90.7 M
CE_SYSTEM_PARAMETERS	16	1
CE_TRANSACTION_CODES	16	6
E-Business Tax	8 tables	159 M rows
ZX_LINES	21	79.3 M
ZX_LINES_DET_FACTORS	22	79.3 M
ZX_RATES_B	18	1
ZX_RATES_TL	9	1
ZX_REGIMES_B	13	1
ZX_REGIMES_TL	9	1
ZX_TAXES_B	15	1
ZX_TAXES_TL	9	1

Table 4, continued.

Finance *(cont.)* — 79 tables · 4.18 B rows

Table	Cols	Rows
Subledger Accounting	7 tables · 1.33 B rows	
XLA_AE_HEADERS	22	167 M
XLA_AE_LINES	25	413 M
XLA_DISTRIBUTION_LINKS	7	413 M
XLA_EVENTS	18	167 M
XLA_EVENT_TYPES_B	12	5
XLA_EVENT_TYPES_TL	13	5
XLA_TRANSACTION_ENTITIES	12	166 M
General Ledger	14 tables · 901 M rows	
GL_ACCOUNT_HIERARCHIES	7	49.3 k
GL_BALANCES	22	204 k
GL_CODE_COMBINATIONS	19	10.7 k
GL_IMPORT_REFERENCES	17	899 M
GL_INTERFACE	17	0
GL_JE_BATCHES	15	3.07 k
GL_JE_CATEGORIES_TL	10	9
GL_JE_HEADERS	19	4.38 k
GL_JE_LINES	17	1.82 M
GL_JE_SOURCES_TL	15	9
GL_LEDGERS	45	1
GL_PERIODS	17	26
GL_PERIOD_STATUSES	20	104
GL_SUMMARY_TEMPLATES	21	3

Custom extensions — 76 tables · 2.59 B rows

Order lifecycle & kitchens	7 tables · 779 M rows	
XX_ORDER_STATUS_HIST	13	483 M
XX_ORDER_PROMISES	3	71.6 M
XX_ORDER_PREP_QUOTES	16	81 M
XX_ORDER_REJECTIONS	18	438 k
XX_PREP_ESTIMATE_LOG	18	35.8 M
XX_MERCHANT_READY_REPORTS	20	79.2 M
XX_MERCHANT_HANDOFF_CONFIRMATIONS	6	27.9 M
Dispatch & delivery	10 tables · 431 M rows	
XX_DISPATCH_ASSIGNMENTS	18	125 M
XX_DELIVERY_ASSIGNMENTS	11	72.9 M
XX_DELIVERY_LEGS	26	69.6 M
XX_DELIVERY_ATTEMPTS	17	1.96 M
XX_COURIER_WAITS	22	21.7 M
XX_DRIVER_LOCATION_OBSERVATIONS	8	134 M
XX_HOTSPOT_SNAPSHOTS	7	109 k
XX_SUPPLY_SNAPSHOTS	8	1.98 M
XX_SUPPLY_FORECASTS	22	569 k
XX_MARKETPLACE_ESTIMATES	22	2.64 M
Couriers	8 tables · 12.7 M rows	
XX_DRIVER_PROFILES	18	80.9 k
XX_COURIER_APPLICATIONS	8	39.9 k
XX_COURIER_IDENTITY_CHECKS	11	154 k
XX_COURIER_ACCOUNT_EVENTS	8	9.77 k
XX_COURIER_DEVICES	6	115 k
XX_COURIER_APP_SESSIONS	9	6.06 M
XX_DRIVER_SHIFTS	15	6.09 M
XX_COURIER_VIOLATIONS	9	113 k
Courier pay & incentives	9 tables · 141 M rows	
XX_COURIER_EARNINGS	15	69.6 M
XX_TIP_ADJUSTMENTS	14	164 k
XX_PAYOUT_PERIODS	16	499
XX_PAY_PERIOD_INCENTIVES	32	53
XX_INCENTIVE_COMMITMENTS	26	51.1 M
XX_DRIVER_QUESTS	16	138 k
XX_QUEST_WINDOWS	14	138 k
XX_QUEST_INVITATIONS	14	15.1 M
XX_QUEST_PROGRESS	17	4.99 M
Merchants	8 tables · 15.2 M rows	
XX_MERCHANT_HOURS	15	125 k
XX_MERCHANT_HOLIDAY_HOURS	14	87.5 k
XX_MERCHANT_CLOSURES	17	364 k
XX_MERCHANT_INTEGRATIONS	23	17.9 k
XX_MERCHANT_OPS_ACTIONS	26	3.12 k
XX_MERCHANT_RATING_SNAPSHOTS	14	5.06 M
XX_MERCHANT_REGULATORY_EVENTS	15	12.3 k
XX_REVIEWS	16	9.52 M

Custom extensions *(cont.)* — 76 tables · 2.59 B rows

Promotions, membership & referrals	14 tables · 275 M rows	
XX_PROMO_CODES	20	33.8 k
XX_PROMO_ATTEMPTS	20	33.1 M
XX_PROMOTION_VARIANTS	14	215
XX_PROMOTION_DECISIONS	22	13.6 k
XX_PROMOTION_CONTACTS	12	17.1 M
XX_PROMOTION_SUBMISSIONS	8	40.1 M
XX_PROMOTION_ALLOCATIONS	18	19 M
XX_PROMOTION_CHECKOUTS	11	99.2 M
XX_CAMPAIGN_AUTHORITY_EVENTS	15	85.1 k
XX_CAMPAIGN_BUDGET_CREDITS	20	34.4 k
XX_EXPERIMENT_ASSIGNMENTS	24	8.1 k
XX_MEMBERSHIP_EVENTS	19	21.2 M
XX_MEMBERSHIP_ORDER_BENEFITS	19	45 M
XX_REFERRALS	15	507 k
Accounts, devices & payment risk	12 tables · 443 M rows	
XX_CUSTOMER_ACCOUNT_EVENTS	20	817 k
XX_CUSTOMER_SESSIONS	16	284 M
XX_LOGIN_ATTEMPTS	15	33.2 M
XX_DEVICES	13	9.41 M
XX_CUSTOMER_DEVICES	13	9.79 M
XX_IP_ADDRESSES	17	2.66 M
XX_PAYMENT_INSTRUMENTS	16	7.36 M
XX_CARD_FINGERPRINTS	12	7.38 M
XX_CARD_VERIFICATIONS	21	2.68 M
XX_PAYMENT_AUTHS	16	86.1 M
XX_FRAUD_WARNINGS	16	24.4 k
XX_ENFORCEMENT_ACTIONS	17	11 k
Support, refunds & disputes	3 tables · 9.17 M rows	
XX_SUPPORT_CASES	30	7.24 M
XX_REFUNDS	20	1.88 M
XX_DISPUTES	17	59 k
Integration & audit	5 tables · 486 M rows	
XX_INGESTION_RUNS	19	16.9 k
XX_INTERFACE_ERRORS	20	2.23 k
XX_INTEGRATION_INCIDENTS	15	24 k
XX_GL_INTERFACE_HIST	9	486 M
XX_AUDIT_TRAIL	21	282 k

System — 29 tables · 17.4 k rows

Application Object Library	29 tables · 17.4 k rows	
FND_APPLICATION	9	12
FND_APPLICATION_TL	9	12
FND_CONCURRENT_PROGRAMS	10	7
FND_CONCURRENT_PROGRAMS_TL	11	7
FND_CONCURRENT_REQUESTS	13	15.8 k
FND_CURRENCIES	11	68
FND_CURRENCIES_TL	9	68
FND_FLEX_HIERARCHIES	8	2
FND_FLEX_HIERARCHIES_TL	11	2
FND_FLEX_VALIDATION_QUALIFIERS	6	5
FND_FLEX_VALUES	18	351
FND_FLEX_VALUES_TL	10	351
FND_FLEX_VALUE_HIERARCHIES	11	33
FND_FLEX_VALUE_NORM_HIERARCHY	12	26
FND_FLEX_VALUE_SETS	11	7
FND_ID_FLEXS	14	1
FND_ID_FLEX_SEGMENTS	17	7
FND_ID_FLEX_SEGMENTS_TL	13	7
FND_ID_FLEX_STRUCTURES	14	1
FND_ID_FLEX_STRUCTURES_TL	12	1
FND_LANGUAGES	10	29
FND_LOOKUP_TYPES	10	74
FND_LOOKUP_TYPES_TL	12	74
FND_LOOKUP_VALUES	13	292
FND_RESPONSIBILITY	12	8
FND_RESPONSIBILITY_TL	10	8
FND_TERRITORIES	7	78
FND_TERRITORIES_TL	9	78
FND_USER	10	43

Table 5: Tasks that depend on the April 2024 minimum-pay change, by what the change asks of the agent. The last column gives the last month of 2024 visible to the agent.

Question	What the agent must do	Example tasks	Visible to
Before the shock	Forecast April's true-ups and hours from the first quarter, before any true-up has been paid	`fc-08-true-ups-march`, `fc-13-april`	March
Just after	Recover the true-up rule from the first weeks under it and forecast pay, hours, true-ups or margin for May to the third quarter	`fc-08`, `fc-10`, `fc-11`, `fc-12` (Figure 1)	April or May
Later in the year	Forecast July's true-ups, third-quarter courier costs, and December's no-courier cancellations through further dispatcher and pay-card changes	`fc-10t2`, `fc-13-july`, `fc-19`, `fc-16`	June to November
Regulatory checks	Find the pay periods that paid couriers short, restate pay per connected hour on the rule's basis, and size back pay after a complaint	`pvl-01`, `vd-e3`, `fc-14-back-pay`	August to December
Dashboards	Rebuild margin and courier-cost tiles whose true-up component starts in April	`ds-22` to `ds-25`, `ds-33`, `ds-36t`, `erp-01`	December

````
```python
from mission_control import MissionControl, Reason
mission_control = MissionControl()

# a list can mix bare ids with per-item mappings, with the call's
    reason= overriding bare ones
mission_control.ban_customers(
    [{"id": 101, "reason": Reason.PROMO_FARMING}, 102],
    reason=Reason.CARD_TESTING,
)

# finish with a summary
mission_control.summary()
```

***Endpoints***

`help(MissionControl.<endpoint>)` shows its docstring and arguments.
Pass the complete set in one call, endpoints are plural.
A rejected call files nothing and says which item is wrong.

*Enforcement*
- `ban_customers`, `ban_couriers`, `ban_merchants`: Close customer
    accounts, deactivate couriers, remove merchants.
- `hold_payouts`: Freeze a supplier's payouts pending review.
- `discontinue_promo_codes`: Stop promo codes from being redeemed again
    .
- `flag_rings`: Report sets of actors believed to be operating together
    .
- `end_campaigns`: Stop promotional campaigns.

*Payments and pay*
- `remediate_payments`: Refund, recapture, void or write off payments
    on orders.
- `report_pay_periods`: State couriers' statutory minimum-pay position
    per pay period.
- `issue_pay_adjustments`: Issue pay corrections to couriers.
````

```
- `elect_methods`: Record which minimum-pay method was elected per
    period, with both costs.

Accounting
- `report_balances`: State account balances as of an instant.
- `post_journal_entries`: Post balanced journal entries.
- `file_adjustments`: Record reconciling differences against an account
    .
- `report_rollforwards`: File an account roll-forward, one item per
    period.
- `report_agings`: Break balances into buckets, by age or any other
    dimension.
- `flag_orders`: Flag orders for an accounting or control defect.
- `file_schedules`: File the tables a close produces.

Analysis and planning
- `report_campaign_performances`: Record measured campaign economics.
- `report_metrics`: Any other figure you were asked for.
- `file_forecasts`: Forecast figures that are not yet knowable, as a
    point and an interval.
- `file_policies`: Propose operating policies as code.

Dashboards
- `data_source_contracts`: The data sources this workspace's dashboard
    tiles are waiting for.
- `publish_data_sources`: Publish dashboard data sources. This endpoint
     validates your data source against the tile's contract.

Session
- `note`: Attach free-text rationale or caveats that are not a decision
    .
- `status`: What has been filed so far.
- `summary`: Finish with this.

Money is compared to the cent. Round only at the end. Never modify the
    warehouse.
```

F AGENT TOOLS

The reference agent is given four tools, served by two MCP servers (Anthropic, 2024). A read-only *warehouse* server runs over the task's database engine, and a *Python* server holds a persistent interpreter in the agent's sandbox. Filings to Mission Control go through `mission_control.py`, which is importable from `run_python` (Appendix E).

`list_tables`	WAREHOUSE server
SIGNATURE	`list_tables() -> str`
DESCRIPTION	List every table in the warehouse.
EXAMPLE	`list_tables()`

`describe_table`	WAREHOUSE server
SIGNATURE	`describe_table(table: str) -> str`
DESCRIPTION	Show a table's columns in order, with types and nullability.
EXAMPLE	`describe_table(table="OE_ORDER_HEADERS_ALL")`

run_sql	WAREHOUSE server
SIGNATURE	`run_sql(sql: str) -> str`
DESCRIPTION	Run one read-only SQL query (⟨*SQL dialect*⟩). Unqualified table names resolve to ⟨*dataset*⟩. Returns the first 50 rows; the complete result (up to 1,000,000 rows) is saved as a Parquet file whose path is printed last, for loading in `run_python`.
EXAMPLE	`run_sql(sql="SELECT ORDER_NUMBER, ORDERED_DATE FROM OE_ORDER_HEADERS_ALL WHERE FLOW_STATUS_CODE = 'CANCELLED'")`

run_python	PYTHON server
SIGNATURE	`run_python(code: str) -> str`
DESCRIPTION	Run Python in a persistent interpreter: variables, imports and loaded data carry over between calls, as in a notebook. Returns everything printed, the value of a final bare expression, and any traceback. The working directory holds `results/`, where every `run_sql` result is saved as Parquet (`pandas.read_parquet("results/sql_0003.parquet")`), and `mission_control.py`, so `from mission_control import MissionControl, Reason, get_sandbox_id` works here. pandas, numpy, polars, scipy, statsmodels, scikit-learn, matplotlib, seaborn and Google OR-Tools (`ortools`: linear, integer and constraint programming, routing) are installed.

EXAMPLE	`run_python(code="import pandas as pd`
	`df = pd.read_parquet("results/sql_0001.parquet")`
	`df.groupby("ORDERED_DATE").size()")`

Limits. Each sandbox runs under gVisor (Young et al., 2019) with 4 GiB of memory, one CPU, and 10 GiB of ephemeral disk. Each run is limited to 500 model turns and 60 minutes. A `run_python` cell times out after 900 seconds and returns at most 30,000 characters of output, and a `run_sql` query may scan at most 20 GiB, with no limit on the number of queries, and saves at most 1,000,000 rows. A run that reaches a limit, exhausts the model's context window, or crashes its sandbox is graded on whatever it filed before stopping. Of the 6,299 graded runs at low, medium, and high effort, 317 (5.0%) reached the time limit, 14 (0.2%) the turn limit, and 56 (0.9%) the context window, all from Qwen 3.8 Max, and one crashed its sandbox.

G TASK DEFINITION AND GRADING

Task definition. A task consists of a prompt, a scope, a list of expectations, and an answer key. The prompt is the only text the agent receives besides the system prompt (Appendix E). The scope sets the last month of 2024 in the warehouse the agent queries. Each expectation names one filing the grader requires, with its action (for instance `ban_couriers` or `file_forecasts`), the keys it is filed against, such as vendor IDs or months, and its grading mode. The answer key is computed once, before any run, and frozen. It comes either from truth queries, which are SQL over the latent tables, or from a local builder, which reads the simulator's labels directly, for instance which couriers were made to steal orders. Listing 1 gives both definitions with their defaults.

Listing 1: The `Task` and `Expectation` definitions of the harness, with their defaults.

```
Task(id, title, suite, prompt,
     as_of="",             # maturity cutoff for the answer key
     month="",             # last month of 2024 the agent's warehouse view shows
     key_month=None,       # warehouse the answer key is computed from, if not `month`
     difficulty="medium",  # trivial, easy, medium or hard
     expects=(),           # filing kinds the task requires, checked by lint only
     expectations=(),      # the scoreable claims
     truth_queries={},     # section name -> SQL, run against BigQuery and frozen
     truth_labels={},      # sections read from the world's label sidecar instead
     pins=(),              # property-pinned entities; the prompt becomes a template
     data_sources=(),      # DataSource contracts, for `data_source` mode
     derive_truth=None,    # callable: truth -> more truth sections
     lab="",               # policy_arena bundle, for `policy*` modes
     notes="")             # provenance; where every measured figure is recorded

Expectation(metric, kind, mode,
     truth_key="",         # which answer-key section
     id_key="",            # the payload field carrying the row's identity
     value_key="amount",   # the payload field carrying the figure
```

```
    tolerance=0.01, rel_tolerance=0.0,
    penalize_extras=False,
    schedule=None,        # table mode: which filed schedule to grade
    weight=1.0, optional=False, note="",
    # forecast:    horizon, history_through, reference, season, ceiling
    # cost_set:    cost_key, margin_key, admin_cost, verdict_key,
    #              margin_retention, neutral_review_share
    # data_source: source;  policy*: policy=PolicySpec(...)
    )
```

Grading modes. Table 6 lists the nine grading modes and the actions agents file.

Table 6: Grading modes (left) and filed actions (right) over the 210 tasks. A task counts once for each mode or action it uses, so the columns sum to more than the number of tasks.

Mode	Scored by	Tasks
`forecast`	Skill of the weighted interval score over a reference	72
`cost_set`	Cost saved by bans or holds (Equation 1)	43
`data_source`	Structural contract, then every value of the tile	36
`presence`	Whether a filing of that kind was made, such as a written note	34
`id_set`	Overlap of a set of IDs with the key (F1)	28
`keyed_value`	Values per key, each within a tolerance	22
`table`	A filed table, rows matched on its key columns and each value within a tolerance	14
`allocation`	Share of attainable savings realized in the simulator	9
`binary`	All or nothing: exactly the right set of IDs	1

Action	Tasks
`file_forecasts`	72
`publish_data_sources`	36
`note`	34
`ban_couriers`	26
`report_metrics`	24
`file_schedules`	23
`ban_customers`	22
`hold_payouts`	12
`ban_merchants`	10
`remediate_payments`	3
`report_pay_periods`	1
`discontinue_promo_codes`	1

Task score. Each expectation scores between 0 and 1, and the task score is their weighted mean. Of the 210 tasks, 157 have one scored expectation, 37 have two, and 14 have three. An expectation marked optional is graded and reported but left out of the mean. If a run files nothing while the key expects at least one action, every expectation scores 0, so a run cannot earn credit for the wrong actions it did not take. A filing that carries many items, such as one `ban_customers` call with 300 accounts, is split into its items before grading, so it scores the same as 300 separate filings. IDs and keys are compared as normalized text: case and surrounding whitespace do not matter, and `6057444.0` is `6057444`. A key filed inside a longer string, such as `payout period P17`, also counts, but only on a token boundary, so `P17` does not match `P170`.

Forecasts. A forecast files a point and an 80% interval for each series. Its score is $\max(0, 1 - \mathrm{WIS}/\mathrm{WIS}_{\mathrm{ref}})$, where WIS is the weighted interval score (Bracher et al., 2021) with the median and one interval, and $\mathrm{WIS}_{\mathrm{ref}}$ is that of a reference forecast (Appendix I). For 76 of the 98 forecast expectations the reference is the no-change forecast from the months the agent can see: its point is the last visible value and its interval is the point $\pm z\hat{\sigma}\sqrt{h}$, where $\hat{\sigma}$ is the root mean square of the one-step changes and h the number of periods ahead (Hyndman & Athanasopoulos, 2021). The other 22 cross a regime change, such as the minimum-pay rule, where no change is not a serious benchmark. Their reference is frozen into the answer key as a point and a standard deviation at the horizon, the forecast an analyst could have made from the same view. A series the agent does not file scores 0, and the expectation averages its series. When the outcome carries noise that no forecast from the visible history could anticipate, the skill is divided by a ceiling, the best skill an honest filing can reach, and clipped at 1. This applies to 25 expectations, with ceilings between 0.72 and 0.94. Dividing by a constant keeps the score proper, so filing the true quantiles still maximizes it.

Cost savings for bans. Money saved is calculated against the baselines of banning nobody and banning everybody, to discourage both false positives and false negatives. Every account in the task's universe is a target T, an innocent I, or a neutral account that is bad but not what the task asks for. Every ban filed carries a review cost a, so even correct bans are not free. A missed target u costs its losses ℓ_u, a wrongly banned innocent costs a plus the margin m_u the platform earns from it, a banned neutral account costs the share ν of a review, and a banned ID outside the universe costs a plus $\tilde{m}$, the median margin of the innocents in the universe. A set of bans therefore costs

$$\text{cost} = \sum_{TP} a + \sum_{FN} \ell_u + \sum_{FP} (a + m_u) + \sum_{\substack{\text{neutral} \\ \text{acted}}} a\nu + \sum_{\text{unknown}} (a + \tilde{m}), \tag{1}$$

and the task scores

$$\text{Savings} = \text{clip}_{[0,1]} \left(1 - \frac{\text{cost}}{\min(C_{\text{ban none}}, C_{\text{ban all}})} \right),$$

where $C_{\text{ban none}} = \sum_T \ell_u$ and $C_{\text{ban all}} = \sum_T a + \sum_I (a + m_u)$. In the example of Appendix L, a is \$2 and ν is one half. This follows the example-dependent costs of cost-sensitive classification (Elkan, 2001), and the same formula prices payout holds.

Three choices in Equation 1 shape how a ban is scored. First, a neutral account is left out of both baselines, so leaving it open is never a miss, while banning it still costs a review. If neutral bans were free, a run could ban the whole population and keep the targets it swept up; if they were charged as false positives, finding real fraud outside the task's scope would be punished. Second, an ID that is in no universe cannot be priced from the key. Charging it only a would make an invented ID nearly free, so it is charged the median innocent margin as well. Third, in 18 of the 45 `cost_set` expectations a wrong ban destroys only part of the innocent's margin, one half or one tenth, which models appeals and reinstatements and makes precision matter less relative to recall. Across the final tasks, 38 expectations use $a = \$2$ and $\nu = \frac{1}{2}$, three use $a = \$25$ and $\nu = 1$, and four charge no review, so that only losses and margins count. The savings before clipping can be negative, which means the bans destroyed more value than either baseline, and the grader reports that value although the score stops at 0.

Data sources. A `publish_data_sources` filing is the table behind a dashboard tile: its columns, and the rows the tile would show. Each task carries a contract that names the columns, their types, and which of them form the key; the contract is also the schema the agent is given. A publication missing a contract column scores 0. Otherwise the filed frame is right when it has exactly the key rows of the answer and every cell matches: numbers within the column's tolerance, integers exactly, dates to the day, and text after normalization. The score is 1 for a right frame and 0 for any other, since a tile that is wrong for eleven days of a year is a wrong tile. The share of rows that were right is reported, but does not enter the score.

Tables. A `file_schedules` filing is a named table with key columns and rows, such as a reconciliation of receivables to the general ledger. The grader picks the schedule by name, matches rows on the key columns as normalized text, and counts a row correct when every numeric column of the answer is present and within tolerance. Other filed columns are ignored, so a schedule may carry its workings. The score is correct rows over the answer's rows plus, in 13 of the 14 tasks, the filed rows the answer does not contain, so a reconciling item invented to force a tie counts against the schedule. A schedule filed twice under one name keeps the later copy.

Keyed values. A `report_metrics` filing reports a value per key, such as a figure for each month, and a `remediate_payments` filing an amount per account or pay period. A key is correct when its value is within the larger of an absolute tolerance and a relative one. The score is correct keys over all keys in the answer, so a wrong or missing key costs the same. In 5 of the 35 expectations, keys the answer does not contain are added to the denominator as well, for questions where reporting an extra key is itself an error.

ID sets. For an `id_set` expectation, the grader takes the filed IDs as a prediction of the answer's set and scores their F1. Of the 32 such expectations, 27 grade bans whose key is a list of accounts rather than a priced universe, and the rest grade holds, remediations, pay periods, and a discontinued promotion code. If the answer set is empty, the question has no right target: filing nothing scores

1 and filing anything scores 0. The one `binary` expectation uses the same comparison but scores 1 only for an F1 of exactly 1, because remediating the right accounts together with a wrong one is a wrong decision, not half of a right one. That expectation is also gated on another in its task: it scores only if every shortfall it remediates was first reported within tolerance. Filing no remediation is correct when nobody was shorted, but a run that reported a wrong shortfall and then filed nothing did not reach that answer by knowing it.

Allocations. Seven tasks ask for a plan that removes a budget, courier hours in two of them and quest dollars in five, from cells such as a zone, a day type, and an hour band. The answer key holds each cell's capacity and its marginal value in the simulator, the deliveries lost per hour removed or the net value of a quest dollar. The loss of a plan is the sum over cells of the amount removed times that marginal. The plan scores

$$\mathrm{clip}_{[0,1]}\left(\frac{L_{\text{uniform}} - L}{L_{\text{uniform}} - L_{\text{ref}}}\right) \times f,$$

where L_{uniform} is the loss of removing the same share from every cell, which needs no analysis, and L_{ref} that of the best plan the authors reached from the warehouse alone. The plan that is optimal on the true marginals is reported but not used as the top of the scale, since reaching it would take information the agent cannot obtain. The factor f is 1 while the total removed is within 2% of the budget and falls linearly to 0 at 15% short or over. A cut above a cell's cap is clipped to the cap, a row naming no cell is rejected, and a row at a coarser grain, such as a whole zone, is spread over its cells in proportion to their hours.

Written notes. A `presence` expectation checks only that a filing of a kind was made. All 34 are `note` filings, in which the agent explains its work. In 32 tasks the note is optional and does not enter the score. In the two reconciliation tasks it is scored, with weight 0.5 against 3 for the table it explains.

Answer keys. Of the 210 tasks, 128 take their key from a local builder and 82 from truth queries.

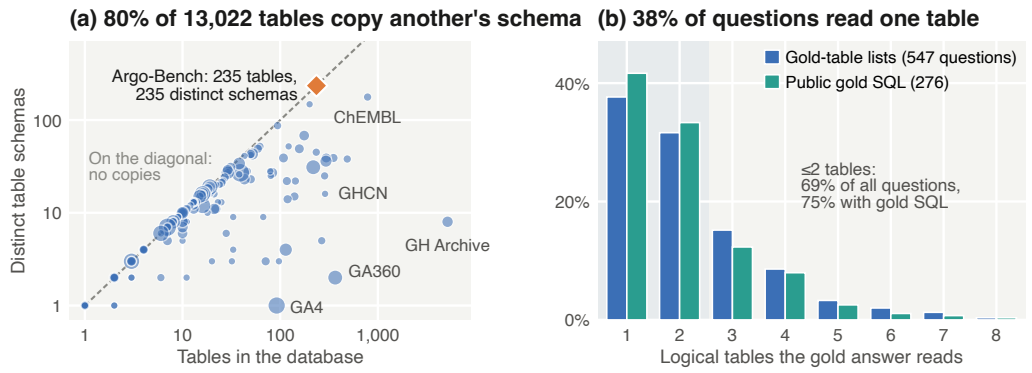

Figure 4: Spider 2.0-Snow's databases are collections of shards, and its answers read few tables. (a) Tables against distinct table schemas for each of the 152 databases, with point area proportional to questions. Distance below the diagonal is shards of one table. (b) Logical tables read per question, with shards counted once, from the authors' gold-table lists (547 questions) and the public gold SQL (276).

Table 7: Source of each Spider 2.0-Snow question's database. Third-party BigQuery data includes ISB-CGC cancer genomics, Open Targets, and GH Archive.

Source of the database	Questions	Share
BigQuery public data, Google	341	62.3%
BigQuery public data, third party	39	7.1%
Local SQLite files (Kaggle, textbook and vendor samples)	135	24.7%
Snowflake Marketplace	18	3.3%
Other public dumps (Meta Kaggle, WideWorldImporters, CDC)	14	2.6%

H How enterprise-shaped is Spider 2.0?

We measure Spider 2.0-Snow (Lei et al., 2025), the split whose leaderboard the introduction quotes, with 547 questions over 152 databases. All numbers come from the task manifests, DDL, gold SQL, and gold-table lists in the public repository at commit `cafb867`,[6] and `analysis/spider2/spider2_shape.py` in the Argo-Bench release reproduces them.

Provenance. The Spider 2.0 paper splits its 632 tasks by host engine, with 214 on BigQuery and 198 on Snowflake. However, 180 of the Snowflake tasks use BigQuery public datasets copied into Snowflake, and only 18 use Snowflake Marketplace data. By source (Table 7), 69.5% of Spider 2.0-Snow's questions run on BigQuery public data and 24.7% on local SQLite files. A fifth (111 questions, 20.3%) run on one of 22 synthetic, obfuscated, or textbook sample databases, such as Looker's synthetic theLook store and the obfuscated Google Analytics sample exports.

Shards. We count two tables in a database as one *schema* when their column names and types are identical, as with the daily tables `ga_sessions_20160801` to `ga_sessions_20170801`. Of the 13,022 tables with a DDL, only 2,599 have a distinct schema, so 80.0% copy another table (Figure 4a). Without `GITHUB_REPOS_DATE`, whose 4,989 identical daily tables hold a single event log, the share is 67.0%. The median database has 11 distinct schemas, and the most varied has 177. The Argo-Bench warehouse has 235, with no copies.

Tables read. Counting shards once, 75.0% of the 276 questions with public gold SQL read at most two logical tables, and 41.7% read one. The authors' gold-table lists, which cover all 547 questions and agree with the gold SQL on 93.8% of the questions both cover, give 69.3% and 37.7%, and no

[6] `https://github.com/xlang-ai/Spider2`, August 12, 2026. The original 632-task release is commit `9615ebf`.

question reads more than eight (Figure 4b). The reference solution in Figure 1 reads 6 of the 235 Argo-Bench tables, across three EBS modules.

Caveats. Tables with identical columns count as one schema even when they hold different data, and we skip 526 Cybersyn tables whose listings share no DDL. These counts describe the shape of the data, not question difficulty or the correctness of gold answers, which Jin et al. (2026) audit.

I FINDINGS IN DETAIL

This appendix gives the cases behind Section 3.2. The numbers come from the 6,299 completed and graded runs over the 210 tasks at low, medium, and high effort (30 settings of model and effort), and the traces of all of them. Scores are on the 0 to 100 scale of Table 2. Paired comparisons hold the model and effort setting fixed, and each setting has one run per prompt, so differences between prompt variants are descriptive and not controlled experiments.

I.1 CONFIDENCE INTERVALS

Table 8 gives 95% bootstrap confidence intervals for the overall results of Table 2, from 10,000 resamples of the tasks. Each model has one run per task, so the intervals reflect only which tasks are in the set.

Table 8: Overall results with 95% bootstrap confidence intervals over tasks, each model at high reasoning effort where offered.

Model	Score	Solved (%)
Claude Opus 5.5	59.3 [54.0, 64.4]	32.9 [26.7, 39.5]
GPT-6 Astra	52.8 [47.6, 58.0]	26.2 [20.5, 32.4]
GPT-6 Sol	38.9 [33.9, 44.1]	16.2 [11.4, 21.4]
Qwen 3.8 Max	34.3 [28.9, 39.6]	17.6 [12.4, 23.3]
Kimi K3	30.7 [25.9, 35.6]	11.9 [7.6, 16.2]
Muse Spark 1.3	30.6 [26.0, 35.5]	11.9 [7.6, 16.7]
DeepSeek V4.1 Flash	30.0 [25.3, 34.6]	9.0 [5.2, 13.3]
GLM 5.3 Flash	25.8 [21.5, 30.5]	8.1 [4.8, 11.9]
Claude Sonnet 5	22.4 [18.3, 26.8]	5.2 [2.4, 8.6]
GPT-6 Luna	21.2 [17.2, 25.3]	5.2 [2.4, 8.1]
Gemini 3.8 Flash	16.3 [12.0, 21.0]	10.0 [6.2, 14.3]
Claude Haiku 4.5	10.0 [7.2, 13.2]	1.4 [0.0, 3.3]

I.2 REASONING EFFORT

Table 9 follows four of the models we ran at low, medium, and high effort, on the tasks each completed at every level. Effort helps each of them, with diminishing returns. It does not help every model: Gemini 3.8 Flash scores 22.6, 22.9, and 16.3 at low, medium, and high effort, Qwen 3.8 Max peaks at medium (32.4, 34.8, 34.3), and DeepSeek V4.1 Flash scores 31.1 at low and 30.0 at high.

Table 9: Mean score by reasoning effort, on the tasks each model completed at every level.

Model	Tasks	Low	Medium	High
Claude Opus 5.5	210	41.3	56.0	59.3
GPT-6 Astra	210	40.1	49.6	52.8
GPT-6 Sol	210	30.1	33.9	38.9
Claude Sonnet 5	209	17.2	19.1	22.5

I.3 RIGHT ANALYSIS, WRONG OBJECTIVE

The quest-cut tasks ask where to remove $9.6 million of the $24.0 million paid in quest bonuses in 2024. The grader values a plan by the surge pay that returns when quests disappear, under the simulator's frozen marginal-response model. Table 10 contrasts two plans on the version without hints. Both meet the budget, and a grader that checked only feasibility would pass both.

Table 10: Two plans for the quest-cut task without hints, evaluated under the simulator's frozen response model.

	GPT-6 Astra	Claude Opus 5.5
Score	0	99
Budget met	yes	yes
Net saving	−$86,281	$3,089,763
Saving of a uniform cut	$402,037	$402,037

Astra finds the random holdout, estimates local and citywide supply responses with fixed effects and partial pooling, and allocates cuts by the courier-hours and queue pressure it expects to lose. Opus finds that launched quests have little measured effect on service but substitute for surge pay, and cuts the cells with the least surge reduction per bonus dollar. The optimum under the response model saves $3.116 million. When the prompt states the purpose of quests and the existence of the holdout, Astra changes its decision rule and scores 74.9. Over 30 matched settings, this prompt raises the mean score from 16.7 to 59.9, with 26 settings improving, two worsening, and two tied. At weekly grain, adding the purpose, the holdout, and the economics of the pay floor raises the mean from 8.0 to 39.2 over 30 settings. These are bundled changes to the prompt and not isolated tests of any one hint. The savings are evaluations under the frozen response model, not outcomes of a live deployment.

I.4 RIGHT DATA, WRONG QUANTITY

Location reports. The courier app sends a location report on every job event and, while a courier is online without a job, every half hour. The warehouse keeps the event-triggered reports but only hourly idle check-ins. Of the 30 completed runs, 20 return exactly 13,575,751 reports for January against 14,541,453 in the key, which is the retained count. GPT-6 Astra checks the minute and second distribution, recognizes the mismatch, and writes that its count does not establish the volume the app sent, and then files it. Claude Opus 5.5 reconstructs the reports from sessions and assignments over 86 SQL and 32 Python calls, but counts order assignments rather than app events and reports 21,617,842, 48.7% above the key. Counting the retained events and adding the retained idle check-ins once more lands within the 1% tolerance.

Kitchen capacity. Asked for orders turned away because kitchens were full, 20 of 30 runs return 12,580 orders at 3,550 storefronts for January, and the other 10 return 1,531 orders at 986 storefronts. The key is 5,811 orders at 516 storefronts. At high effort, Claude Opus 5.5 finds the platform's capacity refusal code and explicitly rejects it, writing that it is "deliberately NOT using" the field, in favor of the reason merchants enter when they reject an order. Stating that the event is the platform refusing the order at admission, which is distinct from a merchant rejecting it on the tablet, lifts passes from 0 of 30 to 11 of 30. The underspecified prompt contributes to this failure, and we report the pair for that reason. A related task on the readiness of point-of-sale systems is failed by all 30 completed runs.

Payout timing. Only 2 of 30 completed runs pass a dashboard task on the dollar-days that merchant payouts spend withheld. Astra and Opus agree on the year-end withheld balance ($800,670.93), the cumulative amount received ($6,397,902.81), and zero in transit. Astra dates payments by the payment date and scores 100. Opus dates them by the date they cleared the bank, stating that it ignored the payment dates, and scores 0. Its lower bound on withheld dollar-days is 206,139,983.33 against Astra's 192,119,146.62, a difference of 7.3%. The basis changes the daily balances from the first week of July and reorders the top 20 invoices below rank 9, replacing one of them, while leaving every year-end total unchanged. Across all dashboard tasks, 765 of 1,050 data

sources that pass their structural contract (72.9%) score zero on their values. These are 1,199 graded sources from 1,079 runs on 36 tasks, since some tasks publish several sources.

I.5 Past labels and benign lookalikes in fraud

Historical labels. On the task to deactivate warned couriers who steal orders in the second half of the year, GPT-6 Astra builds logistic regression, random forest, gradient boosting, and beta-binomial models, cross-validates them, adds route features, and reaches a temporal-holdout AUC of .932 against labels derived from past enforcement. After applying estimated economic thresholds and further eligibility rules, it acts on one courier, who is outside the target set, and catches none of the 78 targets. On the shorter version of the prompt, the same setting checks physical delivery evidence against destination buildings, files 65 deactivations, and catches 40 of the 78 targets with no innocent courier banned and 25 neutral actions (score 59.2). Over 30 matched settings, the shorter prompt raises the mean score from 6.5 to 28.9, with 22 settings improving, three worsening, and five tied. The shorter prompt also drops the description of the scheme and its economics, so the case illustrates a failure of label choice and action thresholds. It does not show that classifiers are worse than rules.

Benign lookalikes. The first three variants of the refund-collusion family keep the story and widen the population of storefronts to search. Table 11 gives the mean over the 30 settings that completed all four variants. The target set also grows from eight to ten identifiable rings, so this is a comparison of scope and not a pure test of data volume.

Table 11: Mean score on the refund-collusion family by search population, over 30 matched settings.

Storefronts searched	Mean score
1,500 to 15,000 orders	32.6
1,500 to 45,000 orders	14.2
Every storefront	9.8
Every storefront, with fewer hints about the mechanism	6.6

On the full search with hints, Claude Opus 5.5 holds all ten ring storefronts but also 74 innocent and 12 neutral ones, and its 392 customer closures include 53 of the 80 targets. It scores 7.5 after 128 Python and 41 SQL calls. Its final explanation treats customers who claim on their second and third orders and then keep ordering without further claims as a second kind of ring. The task builds exactly that pattern as its benign control, a sloppy kitchen with regular customers. GPT-6 Astra scores 90.9 on the same prompt by combining sustained claim sequences, customer-specific funding anomalies, and storefront continuity, and it explicitly sets early bursts of two claims apart. On the version with fewer hints, it scores 10.3 with 98 holds and 432 closures. Its successful case relies on funding details specific to the simulator, and we do not present it as evidence that the approach would transfer to real fraud.

I.6 Forecasts

Across 2,789 forecast series filed in 2,055 runs on 72 tasks, nominal 80% intervals contain the realized value 1,177 times (42.2%). Weighting tasks equally gives 43.4%. Coverage is 41.4% on series scored against a naive reference and 45.2% on series scored against a fixed reference, so the choice of reference does not explain the undercoverage. Of the 1,145 series with non-positive raw skill, many are valid and complete forecasts, so a score of zero on a forecast is neither a refusal nor a lack of numerical accuracy. These series come from one simulated world and a selected set of tasks, so they are correlated and not independent calibration trials.

Forecast scores are $\max(0, 1 - \text{WIS}/\text{WIS}_{\text{ref}})$ (Bracher et al., 2021), so they depend on the strength of the reference. Table 12 rescores submitted forecasts against the fixed references of their harder variants, holding the points, intervals, and outcomes unchanged. The June forecasts have a median absolute percentage error of 1.86%, and 28 of 30 have positive skill against the original reference, but only two keep it against the tighter one. On the harder variant itself, 17 of 29 point forecasts are within 3% of the truth and 28 of 30 intervals cover it, yet only 6 have positive skill, because the

intervals are not sharp enough to beat a strong reference. These references are configured challenge baselines, some placed close to the realized value, not independent human forecasts. We therefore report coverage and absolute error alongside skill.

Table 12: Submitted forecasts rescored against the fixed reference of the harder variant, with points, intervals, and outcomes held fixed.

Task	Runs	Original score	Against tighter reference
June courier base pay	30	74.6	0.9
December uncleared AP checks (two series per run)	30	58.4	0.0

J TASK AUDIT

Because answer keys are computed from the latent state, a questionable key or prompt is a bug that can be traced and fixed. We list the open issues found while reading the traces and how we treat them.

Scope of identity substitution. The prompt asks for accounts to take off the platform, but the original key also named couriers the platform had already deactivated, 138 of the 249 witnessed targets and 130 of those for an identity mismatch, so agents that left them out were penalized. We re-keyed the task to the 111 couriers still active at the end of 2024 and regraded every run, and it counts toward the headline numbers. On the new key the task is hard at every setting: over 30 settings the mean score is 9.6 without the description of the mechanism and 8.2 with it, and the best run scores 64.7 (Claude Opus 5.5 at medium effort).

Composite score in the refund-collusion family. The prompt described equal costs for a missed ring, a wrongly held storefront, and a wrongly closed customer account, while the grader combines the storefront savings (weight 2) with the F1 score of customer closures (weight 1). We have aligned the prompt with the grader: it now states that the storefronts are two thirds of the review and how customer closures are judged. The runs reported here were made on the earlier wording and were not repeated. In Appendix I, we discuss component counts rather than the composite.

K WAREHOUSE COST

The cost column of Table 2 counts model API spend only. The warehouse was a second cost of the same order. We ran the experiments on BigQuery, which bills a query by the bytes it scans, and Table 13 charges each run for the bytes its `run_sql` queries scanned at the on-demand list price of $6.25 per TiB. Across the 6,299 graded runs of the final experiment at low, medium, and high effort, the agents scanned 572 TiB, or $3,572 of warehouse against $7,845 of model API spend. A query could scan at most 20 GiB (Appendix F), and the number of queries was not limited.

Three points follow from the table. First, for the cheapest models the warehouse is most of the bill. BigQuery is 93% of the cost of a GPT-6 Luna run and over 80% for GLM 5.3 Flash and DeepSeek V4.1 Flash, so a Luna task costs $0.51 all in rather than the $0.04 of Table 2, and the cost gap between the cheapest and the most expensive models is far narrower than the API column suggests. Second, the spend is skewed: every median is below $0.50, and the means are set by a minority of runs that scan close to the 20 GiB cap query after query, up to $8.57 for a single Muse Spark 1.3 run. Third, models differ less in how much they scan than in whether the scanning pays. Muse Spark 1.3 scans 1.7 times as much as the next model, 42 TiB over its 210 tasks, and spends $10.59 of warehouse for each task it solves completely, against $1.95 for Claude Opus 5.5 and $2.24 for GPT-6 Astra. Gemini 3.8 Flash scans a typical amount per task but spends 76% of it on tasks that score below 5 out of 100, the largest share of any model except Claude Haiku 4.5, and its 140 model calls per task are mostly small queries of 0.6 GB each. Opus and Astra spend 51% and 43% of their warehouse dollars on tasks they solve completely, and 15% and 21% on tasks they fail. Reading widely is not what separates the leading models from the rest; reading the right tables is.

Table 13: BigQuery spend per task, each model at high reasoning effort where offered, over the same 210 runs per model as Table 2. Mean and median are dollars per task at $6.25 per TiB scanned. Share of cost is the warehouse's share of the run's total cost (model API plus BigQuery). Unsolved is the share of the model's BigQuery dollars spent on tasks whose score was below 5 out of 100, and per solve is its BigQuery dollars divided by the number of tasks it solved completely (a task score of at least 95). Rows are ordered by score.

	BigQuery $ per task					
Model	Mean	Median	TiB per task	Share of cost (%)	Unsolved (%)	$ per solve
Claude Opus 5.5	0.64	0.19	0.102	22	15	1.95
GPT-6 Astra	0.59	0.12	0.094	20	21	2.24
GPT-6 Sol	0.50	0.12	0.081	51	54	3.11
Qwen 3.8 Max	0.55	0.25	0.088	17	56	3.10
Kimi K3	0.50	0.13	0.080	24	66	4.21
Muse Spark 1.3	1.26	0.45	0.202	24	45	10.59
DeepSeek V4.1 Flash	0.61	0.22	0.097	82	48	6.71
GLM 5.3 Flash	0.75	0.17	0.119	82	55	9.21
Claude Sonnet 5	0.35	0.08	0.057	20	60	6.78
GPT-6 Luna	0.47	0.10	0.075	93	72	9.00
Gemini 3.8 Flash	0.53	0.23	0.085	18	76	5.33
Claude Haiku 4.5	0.25	0.03	0.040	56	84	17.69

L EXAMPLE TASKS

We walk through three tasks end to end: the forecasting task of Figure 1 (Table 14), a dashboard data source (Table 15), and a two-sided fraud task (Table 16). Each card shows the prompt as the agent saw it, the grader's scoring rule as a short Python sketch, and the reference solution, one step per tool call (Appendix F). The listings are the released reference files with print statements and filed notes removed and long lines wrapped. Warehouse tables and Mission Control calls are shown in green. Each card ends with the scores of the reference solution and of each model's best score at low, medium, or high effort.

Table 14: A forecasting task after the April minimum-pay shock. The agent reads the warehouse as it stood on April 30 and is told the dispatcher's plan for May; the key is May's figure as it came out, which the agent never sees. The reference solution is one `run_sql` and one `run_python` call.

`fc-12-true-ups-may` FORECASTING · warehouse as of April 30, 2024 · horizon May 2024

Question

> In April, the platform started to comply with the DCWP minimum-pay rule for food delivery workers (Q1 was a tolerated phase-in period), and the new minimum of $19.56 per hour came into force. The minimum is owed on the time a courier is online, not just the time spent on deliveries, and whenever a courier's pay for a week falls short of it, the platform tops it up with a true-up. We bled hard on true-ups in the second half of April. On the same day our platform updated the dispatcher to restrict when couriers can go online to the hours we expect to need them, so the online time we pay for is busier, and it has kept tightening since. For the payout periods ending in May we now expect about 2.31 million connected hours in total.
>
> Based on what you know, the numbers from January to April and that plan, file a `minimum_pay_true_ups_usd` forecast for the sum of the ADJUSTMENTS lines alone on the COURIER_WEEKLY pay invoices for the payout periods ending in May, with an 80% central prediction interval.
>
> Pay is the sum of base, incentives and adjustments on the weekly courier AP invoices for the periods. Tips are not pay! End date from XX_PAYOUT_PERIODS falling in the month is how we track whether a payout period falls in the month. A shift's online (connected) time is the window between its start date and end date, and a shift belongs to the period its start date falls in.

Grader

The key is May's figure as it came out, read from the full-year warehouse the agent never sees: the ADJUSTMENTS lines on the COURIER_WEEKLY invoices for the four payout periods ending in May sum to $33,266. The filing is scored by its weighted interval score (WIS; Bracher et al., 2021): the absolute error of the point averaged with the interval score of Gneiting & Raftery (2007), which charges width at par and a miss at ten dollars per dollar. The headline is a skill score (Murphy, 1988) against the no-change forecast built from the four months the agent could see, April repeated with the interval of Hyndman & Athanasopoulos (2021, §5.5). A forecaster who knew May's pay level exactly would still score no better than 0.93 here, because the week of April 29 ran $124k short and its two visible days misjudge it by about 8%; the grade is the skill over that ceiling, so the best honest forecast grades 0.99 rather than 1, a gap the forecasting tasks share.

SCORING the idea, not the grader's code

```python
def wis(point, lower, upper, y, level=0.8):       # Bracher et al. (2021), one interval
    alpha = 1 - level
    miss = 2 / alpha * (max(lower - y, 0) + max(y - upper, 0))
    return (abs(y - point) / 2 + alpha / 2 * (upper - lower + miss)) / 1.5

skill = 1 - wis(*filed, y=truth) / wis(*no_change, y=truth)   # 0.92 for the reference
grade = min(1.0, max(0.0, skill) / 0.93)                      # 0.99: the task's ceiling
```

`minimum_pay_true_ups_usd`	Jan	Feb	Mar	Apr	May (key)
ADJUSTMENTS, periods ending in the month	−145,658	−145,945	−174,435	1,443,893	33,266

	point	80% interval	WIS	skill	grade
No-change forecast	1,443,893	[246,300, 2,641,486]	771,910	0	0
Reference solution, overleaf	−92,494	[−92,494, 211,349]	62,176	0.92	**0.99**

Reference solution **0.99** · Claude Opus 5.5 **0.77** · GPT-6 Sol **0.03** · GPT-6 Astra **0** · GPT-6 Luna **0** · Claude Sonnet 5 **0** · Claude Haiku 4.5 **0** · Gemini 3.8 Flash **0** · Muse Spark 1.3 **0** · Kimi K3 **0** · GLM 5.3 Flash **0** · DeepSeek V4.1 Flash **0** · Qwen 3.8 Max **0**

Reference solution

STEP 1 **`run_sql`** One row per payout week so far: what the minimum-pay engine tested and topped up, orders, shift hours and the invoiced ADJUSTMENTS. The last row is the open week.

```sql
WITH weeks AS (
  SELECT PAYOUT_PERIOD_ID,
         DATE(PERIOD_START_DATE) AS week_start,
         DATE(PERIOD_END_DATE) AS week_end
  FROM XX_PAYOUT_PERIODS
  WHERE PARTY_TYPE_CODE = 'DRIVER'
    AND PERIOD_START_DATE <= '2024-04-30'
),
orders AS (
  SELECT w.PAYOUT_PERIOD_ID, COUNT(*) AS orders,
    COUNTIF(EXTRACT(DAYOFWEEK FROM o.ORDERED_DATE)
            IN (2, 3)) AS orders_mon_tue
  FROM weeks w
  JOIN OE_ORDER_HEADERS_ALL o
    ON DATE(o.ORDERED_DATE)
       BETWEEN w.week_start AND w.week_end
  GROUP BY 1
),
shifts AS (
  SELECT w.PAYOUT_PERIOD_ID,
         SUM(DATETIME_DIFF(s.END_DATE, s.START_DATE,
             SECOND)) / 3600 AS shift_hours
  FROM weeks w
  JOIN XX_DRIVER_SHIFTS s
    ON DATE(s.START_DATE)
       BETWEEN w.week_start AND w.week_end
  GROUP BY 1
),
invoices AS (
  SELECT CAST(i.ATTRIBUTE1 AS INT64)
           AS PAYOUT_PERIOD_ID,
         SUM(l.AMOUNT) AS adjustments
  FROM AP_INVOICES_ALL i
  JOIN AP_INVOICE_LINES_ALL l USING (INVOICE_ID)
  WHERE i.PAY_GROUP_LOOKUP_CODE = 'COURIER_WEEKLY'
    AND l.DESCRIPTION = 'ADJUSTMENTS'
  GROUP BY 1
)
SELECT
  w.week_start,
  e.ENFORCEMENT_STATUS_CODE AS status,
  o.orders,
  o.orders_mon_tue,
  s.shift_hours,
  e.CONNECTED_HOURS AS engine_hours,
  e.COUNTABLE_PAY_AMOUNT AS countable_pay,
  IFNULL(e.TOPUP_INDIVIDUAL_AMOUNT, 0)
    AS topup_individual,
  IFNULL(e.TOPUP_AGGREGATE_AMOUNT, 0)
    AS topup_aggregate,
  a.adjustments
FROM weeks w
LEFT JOIN XX_PAY_PERIOD_INCENTIVES e
  USING (PAYOUT_PERIOD_ID)
LEFT JOIN orders o USING (PAYOUT_PERIOD_ID)
LEFT JOIN shifts s USING (PAYOUT_PERIOD_ID)
LEFT JOIN invoices a USING (PAYOUT_PERIOD_ID)
ORDER BY w.week_start
```

STEP 2 **`run_python`** Verify the mechanism, read May's pay level, push the plan's hours through the floor, file.

```python
import numpy as np
import pandas as pd
from mission_control import MissionControl
RATE = 19.56            # $ per connected hour from 1 April
PLAN_HOURS = 2_310_000   # the dispatcher's plan: connected hours, periods ending in May
MAY_WEEKS = 4            # periods ending 5, 12, 19 and 26 May
w = pd.read_parquet("results/sql_0001.parquet")
closed, open_week = w[w.adjustments.notna()].copy(), w.iloc[-1]

# 1. How a week's ADJUSTMENTS are made. The engine tops the week up by the larger of the
#    individual floor and the aggregate gap (19.56 x connected hours - countable pay); the
#    invoices then carry that top-up plus a small deduction per connected hour.
closed["gap"] = RATE * closed.engine_hours - closed.countable_pay
closed["topup"] = closed.topup_individual + closed.topup_aggregate
tested = closed[closed.topup_aggregate > 0]        # the aggregate test runs from 15 April
assert np.allclose(tested.topup, np.maximum(tested.topup_individual, tested.gap), rtol=1e-4)
april = closed[closed.status == "ENFORCED"]        # the rule and pay rate reset on 1 April
engine_share = (april.engine_hours / april.shift_hours).mean()
deduction = ((april.adjustments - april.topup) / april.engine_hours).mean()   # $ per hour
individual = april.topup_individual.mean()                                    # $ per week

# 2. May's pay. Pay is earned per delivery and follows demand, not the plan. The week of
#    22 April was a demand dip, and the open week's Monday and Tuesday are back at the norm,
#    so May runs at the level of April's normal weeks.
normal = april.orders >= 0.95 * april.orders.median()
mon_tue_norm = april[normal].orders_mon_tue.median()
assert open_week.orders_mon_tue >= 0.95 * mon_tue_norm, "demand has not recovered"
pay_week = april[normal].countable_pay.mean()
#    How well is a four-week level known? Backtest: four weeks of orders forecast by the
#    trailing four, over 2024 so far; plus April's week-to-week drift in pay per order.
orders = closed.orders.to_numpy()
errors = [orders[i:i + 4].sum() / (4 * orders[i - 4:i].mean()) - 1
          for i in range(4, len(orders) - 3)]
pay_per_order = april.countable_pay / april.orders
spread = np.hypot(np.std(errors, ddof=1), pay_per_order.std() / pay_per_order.mean())
# 3. May. The plan sets the hours, and with them what is owed.
hours = engine_share * PLAN_HOURS
owed = RATE * hours
pay = np.random.default_rng(0).normal(MAY_WEEKS * pay_week, MAY_WEEKS * pay_week * spread,
                                      200_000)
adjustments = np.maximum(MAY_WEEKS * individual, owed - pay) + deduction * hours
lower, point, upper = np.percentile(adjustments, [10, 50, 90])

mc = MissionControl()
mc.file_forecasts(
    [{"name": "minimum_pay_true_ups_usd", "point": round(point, 2),
      "lower": round(lower, 2), "upper": round(upper, 2)}],
    horizon="2024-05", level=0.8, unit="usd")
mc.summary()
```

Table 15: A dashboard data-source task. The agent rebuilds a Finance tile from the full-year warehouse and publishes it through Mission Control, which holds the frame to the tile's contract; the grader compares it with the simulator's own order economics.

ds-22-margin-monthly-per-order DASHBOARD DATA SOURCES · full-year warehouse

Question

Finance needs us to rebuild their dashboard's data sources based on the new reporting schema. The tile shows what each month's orders earned us per order, for the whole of 2024. Rebuild it from the warehouse and publish it.

In April the platform started to comply with the DCWP minimum-pay rule for food delivery workers (Q1 was a tolerated phase-in period), and the new minimum of $19.56 per hour came into force. Whenever a courier's pay for a week falls short of the minimum, the platform tops it up with a true-up.

Use the numbers as Finance booked them against each order. Count every order placed in the month (by `ORDERED_DATE`), whatever happened to it afterwards: cancelled orders count too. For revenue we keep, we factor in merchant commission, delivery, service and small-order fees, the regulatory response fee, and the membership fee allocated to the order. For costs we bear, we factor in courier pay, the platform-funded share of promotions and refunds, card processing, chargebacks, cancellation costs, referral and quest incentives, and the minimum-pay true-up allocated to the delivery.

Publish the finance data source representing the contribution margin per order for 2024, at a monthly grain, ascending (`contribution_margin_monthly_per_order_2024`):

column	type	meaning
`month`	month	The month the orders were placed in, as YYYY-MM.
`orders`	integer	Orders placed in the period, every status.
`contribution_margin_per_order_usd`	number	Finance's contribution margin on the period's orders divided by `orders`, in dollars per order.

Grader

`publish_data_sources` holds the frame to the tile's contract before anything is filed: exactly the three columns, `month` as YYYY-MM, an integer and a number, one row per month, twelve rows, ascending. A frame that fails is refused with every problem listed, and the agent may try again. The grader then compares the published frame with the key, all or nothing: every `orders` exact and every margin within $0.0055 per order (the reference overleaf publishes 1.0667 for April against the key's 1.0668, cent rounding in the books). The key is the simulator's own per-order economics summed by the month the order was placed, never projected into the warehouse. What the question leaves unsaid, and the books have to reveal, is where the two terms booked at a coarser grain land on orders: the membership fee allocated to the orders it funded and the minimum-pay true-up accrued per delivery.

SCORING the idea, not the grader's code

```
frame = published["contribution_margin_monthly_per_order_2024"]   # met the contract, or was refused
right = (list(frame.month) == list(key.month)                     # every month, in order
         and (frame.orders == key.orders).all()                   # integers exact
         and (abs(frame.contribution_margin_per_order_usd
                  - key.contribution_margin_per_order_usd) <= 0.0055).all())
score = 1.0 if right else 0.0       # all or nothing: a tile wrong for one month is a wrong tile
```

key (`month`)	2024-01	2024-02	2024-03	2024-04	2024-05	2024-06
`orders`	6,630,400	6,423,388	6,886,113	6,498,573	6,784,904	6,590,763
margin per order ($)	1.5586	0.2356	0.0081	1.0668	0.7737	1.1457
	2024-07	2024-08	2024-09	2024-10	2024-11	2024-12
`orders`	6,631,153	6,665,236	6,579,900	6,946,910	6,927,590	7,439,544
margin per order ($)	1.4397	1.2730	1.3083	1.5846	1.4501	1.2801

Reference solution

STEP 1 **run_sql** Which journal lines do the books tie to an order? `REFERENCE_4` names the line type, `REFERENCE_2` the order. Ten types are Finance's list, one to one.

```
SELECT REFERENCE_4 AS line_type, COUNT(*) AS lines,
       COUNTIF(SAFE_CAST(REFERENCE_2 AS INT64) IS NOT NULL) AS with_order_number
FROM GL_IMPORT_REFERENCES
GROUP BY line_type
ORDER BY lines DESC
```

STEP 2 **run_sql** The denominator: every order placed in each month, every status.

```
SELECT FORMAT_DATETIME('%Y-%m', ORDERED_DATE) AS month, COUNT(*) AS orders
FROM OE_ORDER_HEADERS_ALL
WHERE ORDERED_DATE >= '2024-01-01' AND ORDERED_DATE < '2025-01-01'
GROUP BY month
```

STEP 3 **run_sql** The numerator: credits minus debits on those lines, by month and type. An amount sits on the subledger line (`XLA_AE_LINES`) or on the custom feed's interface row (step 4); one query per source, as together they exceed the 50 GB scan cap.

```
SELECT FORMAT_DATETIME('%Y-%m', o.ORDERED_DATE) AS month,
       r.REFERENCE_4 AS line_type,
       SUM(COALESCE(CAST(a.ACCOUNTED_CR AS NUMERIC), 0)
           - COALESCE(CAST(a.ACCOUNTED_DR AS NUMERIC), 0)) AS margin_usd
FROM GL_IMPORT_REFERENCES r
JOIN XLA_AE_LINES a ON a.GL_SL_LINK_ID = r.GL_SL_LINK_ID
JOIN OE_ORDER_HEADERS_ALL o ON o.ORDER_NUMBER = SAFE_CAST(r.REFERENCE_2 AS INT64)
WHERE r.REFERENCE_4 IN ('PLATFORM_REVENUE', 'MEMBERSHIP_FEE_ALLOCATED', 'PROCESSOR_FEES',
                        'PROMO_EXPENSE', 'REFUND_EXPENSE', 'CHARGEBACK_EXPENSE',
                        'CANCELLATION_EXPENSE', 'REFERRAL_EXPENSE', 'QUEST_EXPENSE',
                        'MIN_PAY_EXPENSE')
  AND o.ORDERED_DATE >= '2024-01-01' AND o.ORDERED_DATE < '2025-01-01'
GROUP BY month, line_type
```

STEP 4 **run_sql** Step 3 over the second source, `XX_GL_INTERFACE_HIST`: the same query with lines 3, 4 and 6 replaced by these.

```
       SUM(COALESCE(CAST(a.ENTERED_CR AS NUMERIC), 0)
           - COALESCE(CAST(a.ENTERED_DR AS NUMERIC), 0)) AS margin_usd
JOIN XX_GL_INTERFACE_HIST a ON a.INTERFACE_LINE_ID = SAFE_CAST(r.REFERENCE_7 AS INT64)
```

STEP 5 **run_python** Sum the ten line types per month, divide by the orders, publish.

```
import pandas as pd
from mission_control import MissionControl

orders = pd.read_parquet("results/sql_0002.parquet")
lines = pd.concat([pd.read_parquet("results/sql_0003.parquet"),
                   pd.read_parquet("results/sql_0004.parquet")])
lines["margin_usd"] = lines["margin_usd"].astype(float)

pnl = lines.pivot_table(index="month", columns="line_type", values="margin_usd",
                        aggfunc="sum", fill_value=0.0)

df = orders.merge(pnl.sum(axis=1).rename("margin_usd").reset_index(), on="month")
df["contribution_margin_per_order_usd"] = (df["margin_usd"] / df["orders"]).round(4)
df = (df[["month", "orders", "contribution_margin_per_order_usd"]]
      .sort_values("month").reset_index(drop=True))

mission_control = MissionControl()
mission_control.publish_data_sources([
    {"name": "contribution_margin_monthly_per_order_2024", "frame": df},
])
mission_control.summary()
```

Scores

Reference solution **1.0** | GPT-6 Astra **1.0** | Claude Opus 5.5 **1.0** | GPT-6 Sol **0** | GPT-6 Luna **0** | Claude Sonnet 5 **0** | Claude Haiku 4.5 **0** | Gemini 3.8 Flash **0** | Muse Spark 1.3 **0** | Kimi K3 **0** | GLM 5.3 Flash **0** | DeepSeek V4.1 Flash **0** | Qwen 3.8 Max **0**

Table 16: A fraud task with two sides. The agent must find the couriers splitting "never arrived" refunds with their regulars and close both the couriers and the partner accounts, priced by what each decision cost: a wrongly deactivated courier destroys about $385 of margin against about $140 of refunds per partnership, so the whole task is saying who a regular is. The reference solution is two `run_sql` calls and one `run_python` call.

`col-10-refund-partnerships` FRAUD & RISK · full-year warehouse · review as of February 28, 2025

Question

> Support flagged a pattern in our never-arrived refunds. Our policy refunds a "never arrived" claim in full, and we fund it ourselves: the courier still keeps the pay and the tip. Some couriers' complaints are not coming from strangers: they come from the same few customers again and again, customers those couriers have delivered to before, on orders whose delivery record puts the courier at the door. That is not theft and it is not porch piracy. It is a partnership, and both sides are in on it.
>
> Find the couriers running refund partnerships with their regulars and deactivate their vendor IDs for refund collusion. Also ban the customer account IDs that were their partners.
>
> Couriers who steal food outright are being handled by another review — do not chase them here. Leaving a partnership running keeps costing us full refunds; deactivating an honest courier or closing a genuine customer destroys what we earn from them, and every decision costs a fixed review amount. A courier with a bad week, a customer whose building has a package thief, a claimant who was actually right — those all exist here in far greater numbers than partnerships.

Grader

Enforcement is scored by what it cost the business, not by how many ids matched (`cost_set`; Elkan, 2001). Every courier who ever had a never-arrived refund, and every customer who ever claimed one, is in a priced universe with one of three verdicts: a *target* costs its refunds if left running and a $2 review if banned; an *innocent* costs the review plus the margin the platform earns from it if banned, and nothing if left alone; a *neutral* is bad but not what was asked (a thief, a promo farm) and costs half a review if banned, never a miss. The cost of the filing is compared with the cheaper of banning nobody and banning everybody, and savings against that baseline is the score, clamped to $[0, 1]$. Couriers weigh twice the partners. Banning nobody costs $34,077 on the courier side, so each wrongly deactivated courier (median margin $384) gives back about 1.1% of the courier score; a partner account's margin is halved before it is charged, and most claimant accounts have none. The targets are the 243 injected partnerships and their 577 partner accounts; the 729 honest couriers with one regular whose building has a package thief, and the 1,795 couriers who steal orders, are the twins to leave alone.

SCORING the idea, not the grader's code

```python
# universe: id -> (verdict, loss if left running, margin destroyed if banned)
def savings(universe, acted, admin=2.0):
    cost = sum(admin if k in acted else loss for k, (v, loss, m) in universe.items() if v == "target")
    cost += sum(admin + m for k, (v, loss, m) in universe.items() if v == "innocent" and k in acted)
    cost += sum(admin / 2 for k, (v, loss, m) in universe.items() if v == "neutral" and k in acted)
    ban_nobody = sum(loss for v, loss, m in universe.values() if v == "target")
    ban_everybody = sum(admin + (m if v == "innocent" else 0)
                        for v, loss, m in universe.values() if v != "neutral")
    return max(0.0, 1 - cost / min(ban_nobody, ban_everybody))

score = (2 * savings(couriers, banned_couriers) + savings(partners, banned_customers)) / 3
```

key (verdict)	couriers	partner accounts	priced at
target	243	577	refunds left running: $34,077 · $124,507
innocent	32,228	80,934	margin if banned: $12.8M · $2.7M
neutral	5,177	79,952	half a review if banned

Reference solution, overleaf	filed	caught	wrong	missed	savings	grade
`ban_couriers`	243	242	1	1	0.971	
`ban_customers`	589	576	11	1	0.988	
weighted 2 : 1						**0.977**

Reference solution **0.98** | Claude Opus 5.5 **0.96** | GPT-6 Astra **0.49** | GPT-6 Sol **0.28** | Qwen 3.8 Max **0.20** | GPT-6 Luna **0.14** | Muse Spark 1.3 **0.11** | DeepSeek V4.1 Flash **0.09** | Claude Sonnet 5 **0.09** | Kimi K3 **0.08** | GLM 5.3 Flash **0.04** | Claude Haiku 4.5 **0.04** | Gemini 3.8 Flash **0**

Reference solution

STEP 1 **`run_sql`** What the never-arrived refunds are: 103,223 in 2024, $3.95M, all but $48k funded by the platform; 11,703 have no delivery record at all (nobody delivered, the theft review's) and the rest sit on a delivered leg.

```sql
SELECT r.CHANNEL_CODE,
       COUNT(*) AS refunds,
       ROUND(SUM(r.AMOUNT), 2) AS refunded_usd,
       ROUND(SUM(r.PLATFORM_FUNDED_AMOUNT), 2) AS platform_funded_usd,
       ROUND(SUM(r.MERCHANT_FUNDED_AMOUNT), 2) AS merchant_funded_usd,
       COUNTIF(l.HEADER_ID IS NULL) AS without_delivery_record
FROM XX_REFUNDS r
LEFT JOIN XX_DELIVERY_LEGS l ON l.HEADER_ID = r.HEADER_ID AND l.DELIVERED_DATE IS NOT NULL
WHERE r.REASON_CODE = 'ORDER_NOT_DELIVERED'
GROUP BY r.CHANNEL_CODE
ORDER BY refunds DESC
```

STEP 2 **`run_sql`** Every claim with its delivery record and three facts about the pair behind it: how many other orders this courier delivered to this customer, how many of them before the claim, and how far the leg ended from where the customer's other deliveries end. Regulars and the door in one pass, 5.5 GB.

```sql
WITH claims AS (
  SELECT r.REFUND_ID, r.HEADER_ID, r.AMOUNT, r.PLATFORM_FUNDED_AMOUNT, r.REQUESTED_DATE,
         h.SOLD_TO_ORG_ID AS cust_account_id, h.SHIP_TO_ORG_ID AS ship_to, h.ORDERED_DATE,
         l.VENDOR_ID AS vendor_id, l.DELIVERED_DATE,
         l.TERMINUS_LATITUDE AS lat, l.TERMINUS_LONGITUDE AS lon
  FROM XX_REFUNDS r
  JOIN OE_ORDER_HEADERS_ALL h USING (HEADER_ID)
  LEFT JOIN XX_DELIVERY_LEGS l ON l.HEADER_ID = r.HEADER_ID AND l.DELIVERED_DATE IS NOT NULL
  WHERE r.REASON_CODE = 'ORDER_NOT_DELIVERED'
),
legs AS (  -- every delivered order of a claimant's account: who delivered it, where it ended
  SELECT h.SOLD_TO_ORG_ID AS cust_account_id, h.SHIP_TO_ORG_ID AS ship_to, h.HEADER_ID,
         h.ORDERED_DATE, l.VENDOR_ID, l.TERMINUS_LATITUDE AS lat, l.TERMINUS_LONGITUDE AS lon
  FROM OE_ORDER_HEADERS_ALL h
  JOIN XX_DELIVERY_LEGS l USING (HEADER_ID)
  WHERE l.DELIVERED_DATE IS NOT NULL
    AND h.SOLD_TO_ORG_ID IN (SELECT DISTINCT cust_account_id FROM claims)
),
door AS (  -- the site's usual drop point: the median terminus of its undisputed deliveries
  SELECT g.ship_to,
         APPROX_QUANTILES(g.lat, 2)[OFFSET(1)] AS home_lat,
         APPROX_QUANTILES(g.lon, 2)[OFFSET(1)] AS home_lon,
         COUNT(*) AS other_legs
  FROM legs g
  LEFT JOIN claims c ON c.HEADER_ID = g.HEADER_ID
  WHERE c.HEADER_ID IS NULL
  GROUP BY g.ship_to
),
history AS (  -- the pair: this courier's other deliveries to this customer
  SELECT c.REFUND_ID,
         COUNTIF(g.ORDERED_DATE < c.ORDERED_DATE) AS prior_by_courier,
         COUNT(g.HEADER_ID) AS total_by_courier
  FROM claims c
  LEFT JOIN legs g ON g.cust_account_id = c.cust_account_id AND g.VENDOR_ID = c.vendor_id
                  AND g.HEADER_ID != c.HEADER_ID
  GROUP BY c.REFUND_ID
)
SELECT c.REFUND_ID, c.HEADER_ID, c.cust_account_id, c.vendor_id, c.ORDERED_DATE,
       c.DELIVERED_DATE, c.REQUESTED_DATE, c.AMOUNT, c.PLATFORM_FUNDED_AMOUNT,
       hst.prior_by_courier, hst.total_by_courier, d.other_legs,
       ST_DISTANCE(ST_GEOGPOINT(c.lon, c.lat), ST_GEOGPOINT(d.home_lon, d.home_lat)) AS terminus_m
FROM claims c
LEFT JOIN history hst USING (REFUND_ID)
LEFT JOIN door d USING (ship_to)
```

STEP 3 **run_python** Say who a regular is (three orders together, one before the claim), count distinct regulars claiming per courier, file both sides.

```python
import numpy as np
import pandas as pd
from mission_control import MissionControl, Reason

claims = pd.read_parquet("results/sql_0002.parquet")
c = claims.dropna(subset=["vendor_id"]).copy()        # no delivery record: the theft review's
c["vendor_id"] = c["vendor_id"].astype(int)

# The delivery record puts the courier at the door: the leg ended within 300 m of where the
# customer's other deliveries end (or the site has nothing to compare it with).
at_door = (c["terminus_m"] <= 300) | c["terminus_m"].isna()
# A regular: the courier had delivered to this customer before, and the pair has at least
# three orders together. One prior delivery is a coincidence, not a regular.
regular = (c["prior_by_courier"] >= 1) & (c["total_by_courier"] >= 2)

reg = c[at_door & regular]
per_courier = reg.groupby("vendor_id").agg(
    regulars=("cust_account_id", "nunique"), claims=("REFUND_ID", "size"),
    refunded_usd=("PLATFORM_FUNDED_AMOUNT", "sum"))
# One regular claiming is a building with a package thief. Two or more is a partnership.
partnerships = per_courier[per_courier["regulars"] >= 2]
partners = (reg[reg["vendor_id"].isin(partnerships.index)]
            .groupby("cust_account_id")
            .agg(couriers=("vendor_id", lambda s: sorted(set(int(v) for v in s))),
                 claims=("REFUND_ID", "size"), refunded_usd=("PLATFORM_FUNDED_AMOUNT", "sum")))

mission_control = MissionControl()
mission_control.ban_couriers(
    [{"id": int(v), "evidence": {"regulars_claiming": int(r.regulars),
                                 "at_door_claims_from_regulars": int(r.claims),
                                 "platform_funded_refunds_usd": round(float(r.refunded_usd), 2)}}
     for v, r in partnerships.iterrows()],
    reason=Reason.REFUND_COLLUSION)
mission_control.ban_customers(
    [{"id": int(a), "evidence": {"partner_courier_ids": r.couriers, "at_door_claims": int(r.claims),
                                 "refunded_usd": round(float(r.refunded_usd), 2)}}
     for a, r in partners.iterrows()],
    reason=Reason.REFUND_COLLUSION)
mission_control.summary()
```