# FROM SOLVING DATA TASKS TO LEARNING A DATA SPACE: BENCHMARKING LEARNING CAPABILITIES OF DATA AGENTS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Data agents are increasingly deployed in enterprise analytical platforms, where they serve sustained workloads of queries over the data space. These workloads often contain queries that share procedures, depend on domain knowledge specific to the data space, and are affected by changes in it. Without learning capabilities, data agents may repeat costly exploration for each query, miss knowledge they have already found, and produce wrong answers from outdated experience. Existing data agent benchmarks evaluate each query independently and do not measure the learning capabilities of data agents. In this work, we introduce DataIntern, a benchmark that evaluates the *learning capabilities* of data agents through: (1) 15 real-world enterprise-level data spaces and 742 queries organized into 84 workloads that cover three complementary learning scenarios: *procedural reuse*, *data space understanding*, and *data space changes*; (2) a feedback-aware agentic construction framework, in which real enterprise feedback guides agents to build data spaces from public data and to verify by execution that each query needs this domain knowledge; (3) a controlled evaluation protocol that evaluates the learning capabilities of data agents under different experience conditions and clearly reveals how much they gain from experience and at what cost; (4) extensive experiments across thirteen frontier LLMs, six agent harnesses, and four memory mechanisms that reveal key insights for building future data agents with stronger learning capabilities. Our results show that reusing procedures cuts tool calls by 81 % and cost by 60 % on DataIntern-Full. Compared with unrelated experience from the same data space, experience that contains this domain knowledge raises accuracy by 15–28 points for twelve of thirteen models on DataIntern-Lite. Data space changes remain the hardest scenario. Few data agents answer correctly before and after a change, and most failures come from applying knowledge where it does not hold. Code and data are available at `https://anonymous.4open.science/r/DataIntern-E53C`.

## 1 INTRODUCTION

Data agents are playing an increasingly important role in enterprise analytical platforms such as Databricks and Snowflake (Databricks, 2025; Snowflake, 2025), where they answer natural-language queries over databases, files, and documents. Recent data agent methods (Vaidya et al., 2025; Biswal et al., 2026; Agarwal et al., 2026) based on large language models (LLMs) have made rapid progress on established benchmarks such as BIRD (Li et al., 2023), Spider 2.0 (Lei et al., 2025), and DAComp (Lei et al., 2026), and on a growing set of benchmarks for broader data agent tasks (Huo et al., 2026; Jing et al., 2025; Lai et al., 2026; Li et al., 2026). These benchmarks evaluate whether a data agent can solve individual analytical queries over heterogeneous data, covering tasks from text-to-SQL to end-to-end data engineering and open-ended analysis.

In practice, data agents serve sustained workloads over the same enterprise data space, often spanning weeks or months. Take the US retail banking supervision data space as an example (Figure 1). It holds the quarterly financial reports of US banks, consumer complaints about these banks, and thousands of government notices on banking policy. A supervision team uses it to review bank finances and the policies that apply to banks. Its workload poses three learning challenges. Some



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.

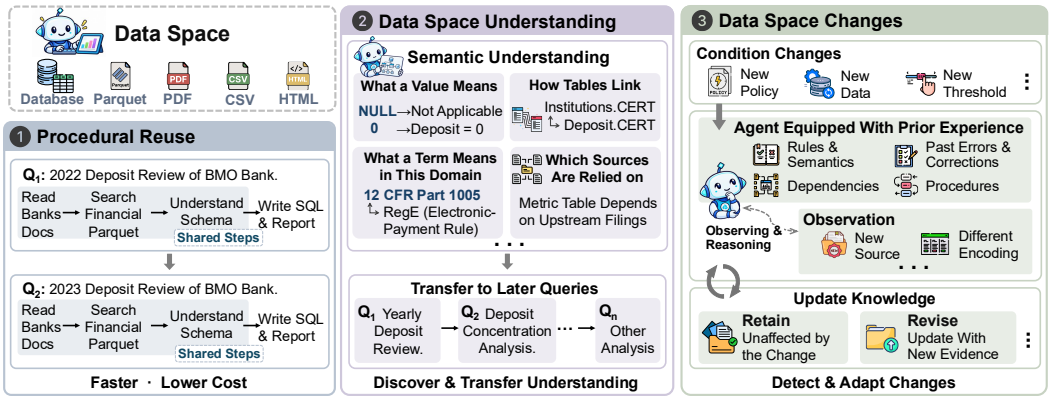



Figure 1: Three learning scenarios over data space.

queries repeat an earlier procedure (**Procedural Reuse**), e.g., a deposit review of one bank for 2022 and the same review for 2023. After the first review, a data agent can do the second with far fewer steps. Some queries depend on the data semantics of the data space, such as column meanings, value encodings, and source dependencies (**Data Space Understanding**). For example, analysts call a payment rule *Reg E*, but the documents in the data space name it only by its legal code, *12 CFR part 1005*. A data agent that has learned this link can find the right documents for a later query about *Reg E*. Some queries follow a change of conditions, such as a new policy, and old knowledge may no longer apply (**Data Space Changes**). For example, banks below a size threshold are exempt from a federal mortgage rule, and a new notice raises that threshold from $11.8 billion to $12.2 billion. A data agent that has recorded both figures can apply $12.2 billion to a current query and $11.8 billion to a report filed under the old notice.

Data agents face these challenges in real workloads, but existing benchmarks cannot yet evaluate them for two reasons (Table 6; §5). First, most data agent benchmarks are designed to evaluate how well a data agent solves **static, independent queries**, each from a fresh state. Multi-turn benchmarks such as BIRD-Interact (Huo et al., 2026) place related queries in one session. They test whether a data agent can ask the user for clarification and answer follow-up queries, but not whether it learns from its own experience over a sustained workload. Second, no existing benchmark measures the **learning capabilities of data agents** over heterogeneous enterprise data spaces. Continual learning benchmarks (Asawa et al., 2026; Wang et al., 2026a) have begun to evaluate the learning capabilities of agents. For data agent tasks, however, they cover only limited modalities and task types, e.g., single-modality databases, far from real enterprise data spaces and workloads. At the same time, modern agent harnesses and memory systems (Park et al., 2023; Packer et al., 2023; Chhikara et al., 2025; Xu et al., 2025; Fu et al., 2026) have begun to give data agents the ability to learn continually, which makes it both possible and timely to measure these capabilities.

To fill this gap, we present DataIntern, a benchmark for evaluating learning capabilities of data agents over enterprise data spaces. DataIntern provides 15 real-world enterprise-level data spaces and 742 queries organized into 84 workloads. The workloads cover the three learning scenarios above, and a controlled evaluation protocol measures the learning capabilities of data agents. Building such a benchmark raises two challenges. *On the construction side*, each query must depend on domain knowledge that is specific to the data space and cannot be inferred from general knowledge acquired during pre-training. We call such domain knowledge *data space knowledge*. It can be the meaning of a column or value, a procedure that its analysts follow, or an industry term with a specific meaning in the data space. We therefore design a feedback-aware agentic construction framework, in which real enterprise feedback guides agents to build every data space from public data and verify by execution that each query depends on its data space knowledge. *On the evaluation side*, learning capabilities are hard to measure from a single run. We therefore design a controlled evaluation protocol that evaluates the same query under different experience conditions and clearly reveals how much data agents gain from experience and at what cost.

We make the following key contributions: (1) **New evaluation problem**: We formalize the problem of evaluating learning capabilities of data agents, covering *procedural reuse*, *data space understanding*, and *data space changes*. (2) **Challenging benchmark**: We introduce DataIntern with 15

real-world enterprise-level data spaces and 742 queries organized into 84 workloads, together with a controlled evaluation protocol that evaluates learning capabilities under different experience conditions. (3) **Feedback-aware agentic construction framework**: Real enterprise feedback guides agents to build data spaces from public data, synthesize queries and workloads, and verify by execution that each query needs its data space knowledge. (4) **Extensive experiments and insights**: We evaluate thirteen frontier LLMs, six agent harnesses, and four memory mechanisms, and derive guidelines for future data agent development.

Our experiments lead to three main findings. First, reusing procedures cuts tool calls by 81 % and cost by 60 % on DataIntern-Full. Second, compared with unrelated experience from the same data space, experience that contains the data space knowledge raises accuracy by 15–28 points for twelve of thirteen models on DataIntern-Lite. Third, data space changes remain the hardest scenario. Few data agents answer correctly before and after a change, and most failures come from applying knowledge where it does not hold.

## 2 Problem Definition

**Task definition.** A data agent $A$ operates over a *data space* $W$ of databases, data files (e.g., CSV and Parquet), and documents (e.g., regulations and reports in PDF or HTML). It receives a *workload* $(q_1, \ldots, q_n)$ of natural-language analytical queries in order. Between queries, $A$ maintains a persistent state $S$ (notes, code, rules, or any form it chooses) and answers each query $q$ with

$$y = A(W, q; S_H), \tag{1}$$

where $S_H$ is the state after completing the earlier queries $H$. The benchmark measures whether $S_H$ improves the accuracy of $y$ or reduces the cost of producing it. A *workload* is a sequence of queries over one data space. A *test* consists of *source queries* that give the data agent experience and *target queries* that are scored under the settings of Table 1. The remaining queries are *background queries* from the same data space. We study three learning scenarios (Figure 1). In **procedural reuse**, a later query repeats a similar analysis. In **data space understanding**, a query needs data space knowledge. The data agent can gain this knowledge by exploring the data space itself or while working on other queries. In **data space changes**, some condition in the data space has changed, e.g., a reporting threshold is raised or a new data source follows different rules. The data agent is tested on whether it uses the proper knowledge for each condition.

**Evaluation protocol.** Learning capabilities are hard to measure from a single run. We therefore control what a data agent carries into a query and compare the results (Table 1). For procedural reuse, the data agent solves a *reuse chain* of three similar queries in one session. For data space understanding, each target query starts a new session under four settings (C0–C3). C1 and C2 each carry experience from the same number of queries in the same data space, but only the C2 queries contain the knowledge. C1 therefore serves as a *matched control*, and the gain of C2 over C1 measures the effect of the knowledge. For data space changes, each test has a *change point*, the first query where a condition has changed, such as a new policy or a new data release. The data agent builds experience on the queries before it. The four settings (B0–B3) differ in whether the data agent solves the query at the change point and how it keeps what it learns there. Under each setting, the data agent then answers the queries after the change and one query before the change.

Table 1: Evaluation settings.

| | |
|---|---|
| *Data space understanding (new session)* | |
| C0 | No experience |
| C1 | Unrelated experience (same number of queries, similar interaction) |
| C2 | Experience that contains the knowledge |
| C3 | C1 plus the knowledge stated (oracle) |
| *Data space changes (after the change point)* | |
| B0 | Skips the query at the change point |
| B1 | Solves it and keeps session logs |
| B2 | Solves it and updates its memory |
| B3 | Is told the new knowledge (oracle) |

**Evaluation metrics.** A query is *solved* when every deterministic grading check passes, and accuracy is the share of solved queries. **(1) Procedural reuse**: accuracy on the first and third queries (Acc@1, Acc@3), and the tool calls of the third query relative to the first (Tool@3). **(2) Data space understanding**: accuracy under C0 (Cold) and C2 (Use); the learning gain $\mathrm{LG} = \mathrm{C2} - \mathrm{C1}$, the

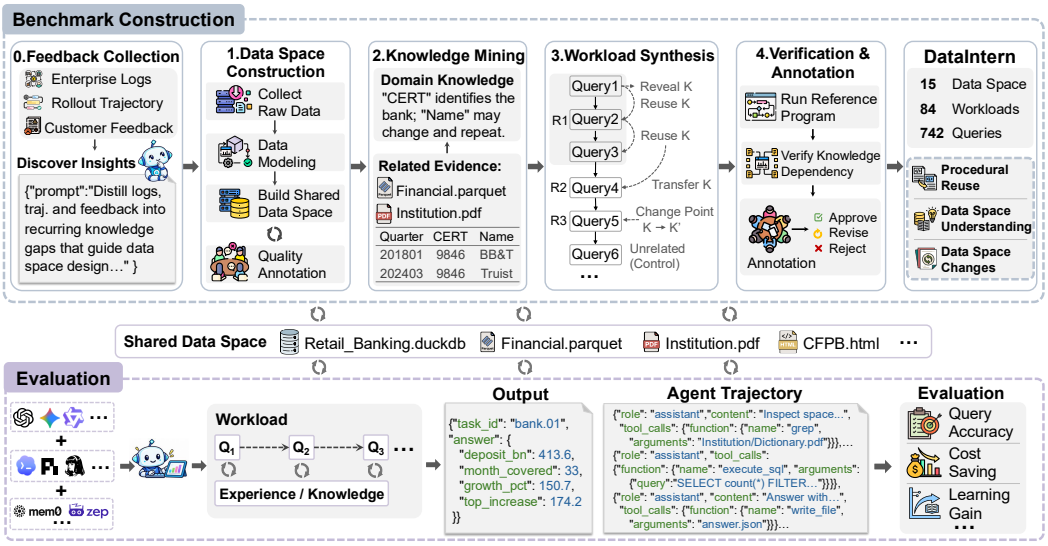



Figure 2: Overview of DataIntern.

gain from relevant experience; the gap closure $\mathrm{Gap} = (\mathrm{C2} - \mathrm{C1})/(\mathrm{C3} - \mathrm{C1})$, the fraction of the oracle gain that experience recovers; and the negative transfer rate (Neg.), how often relevant experience lowers accuracy. **(3) Data space changes**: accuracy after the change point (New), how often answers still use the outdated knowledge (Recur), accuracy on the query before the change point (Retain), and the share of tests where the queries after the change point and the query before it are all correct (Joint). **(4) Cost**: the input, cached input, and output tokens, the tool calls, and the wall time of every scored query, including reading and retrieving experience. We report the cost in USD at list price per query ($/q) and per solved query ($/solve), and the experience tokens (Exp. tokens).

## 3 Benchmark Construction

This section details how DataIntern is built (Figure 2). We first describe how we construct 15 data spaces from public data guided by enterprise feedback (§3.1), then explain how queries and workloads are synthesized over them (§3.2), and finally show how every query is verified by execution and by experts (§3.3), following the design principles in Appendix G.

### 3.1 Data Space Construction

To reflect real enterprise settings, we build every data space from public data in the same domains as enterprise workloads. Public data already contains what makes enterprise data hard to use, such as links between tables and documents, rules written only in documents, and conditions that change with a new policy or data release. For example, in the US retail banking supervision data, banks may rename following mergers, and 228 bank names are each shared by several banks. The financial tables therefore identify a bank by its CERT number, a unique bank ID that the table definitions do not document (Figure 1). A data agent that groups records by bank name will mix up different banks. In this way, each data space stays close to a real enterprise environment in topic, scope, and scale.

**Enterprise feedback.** To guide data space construction, we collect feedback from enterprise execution logs, agent trajectories, and customer reports. Construction agents cluster the feedback. Eight PhD-level experts in agents and databases then turn the clusters into requirements for the data spaces, namely which kinds of data space knowledge the queries should need and how requests recur in a workload. Construction agents then gather public data that contains such knowledge. The feedback is anonymized and released with the benchmark (Appendix H).

**Data collection.** Construction agents collect raw data from open public sources in areas such as finance, energy, healthcare, education, and advertising. The agents parse and normalize the data into

databases and files, while keeping the original conventions of each source. For example, an empty deposit value in the banking data means that the item does not apply, while a zero value means that the bank holds no such deposits (Figure 1). Each data space contains 13–68 tables, 56–1,150 data files, and 44–5,149 documents (Table 27 in Appendix F).

### 3.2 Query and Workload Synthesis

To control which data space knowledge each query depends on, we synthesize queries with known dependencies. An LLM-based authoring agent writes each query in the requester's voice against the real data. The query names the business metric and the deliverable but does not reveal the rules of the data space. For each query, the agent also produces: **(1)** a reference program that computes every answer field from the data at run time, **(2)** deterministic grading checks, and **(3)** a rubric of six to ten items. When a query depends on a piece of data space knowledge, the reference program also computes the answer obtained without it (the *wrong answer*). For example, a query asks how many banks are exempt from mortgage reporting under the 2025 policy. The reference program returns 315 with the threshold of $58 million and the wrong answer, 220, with the old threshold of $47 million. The difference confirms that the answer depends on using the threshold knowledge.

We then organize the queries into workloads. A workload is a sequence of queries over one data space and contains several tests. In each test, *source queries* give the data agent a chance to learn a piece of data space knowledge, and *target queries* need the same knowledge in a different analysis. For example, a source query can show that banks are identified by their CERT numbers, and a later target query about the banks named *Peoples Bank* needs this rule. Only target queries are scored, and each comes after its source queries. The other queries in the workload are *background queries*, ordinary requests on the same data space that are not scored. Every query can be solved from the data space alone and never needs the answer of an earlier query. For data space changes, we build tests around condition changes that the data already contain (Appendix I).

### 3.3 Verification and Expert Annotation

Before release, we check each reference program and each test by execution, and experts review each query.

Table 2: Statistics of DataIntern.

| *Shared across both tiers* | | |
|---|---|---|
| Data spaces | | 15 |
| Tables and views | | 450 |
| Documents | | 13.6k |
| Data volume (GB) | | 68 |
| | **Full** | **Lite** |
| Queries | 742 | 193 |
| Procedural reuse | 175 | 73 |
| Data space understanding | 170 | 72 |
| Data space changes | 110 | 48 |
| Background and controls | 287 | 0 |
| Knowledge units evaluated | 83 | 35 |
| Words per request | 44 | 46 |
| Answer fields per query | 5.1 | 5.3 |
| Checks and rubric items per query | 12.3 | 13.4 |
| Cold-start pass rate | 0.63 | 0.69 |

**Reference program verification.** Every reference program is executed twice and must return the same answer. Before the release was frozen, an independent agent also solved each query without seeing the reference. Its answers agree with the reference on 601 of the 742 queries, and every disagreement was adjudicated, with the reference program fixed when it was wrong (Appendix I).

**Test verification.** A candidate test (a source query, a piece of knowledge, and a target query) is released only when three execution-based checks pass: **(1)** the reference program of the source query reads the tables or documents that hold the knowledge, which we verify by tracing what it reads; **(2)** the target query needs the knowledge, which we check with its wrong answer and with runs where the knowledge is stated directly; **(3)** a data agent without experience still fails or needs many steps on the target query. For each target query, we also select unrelated queries from the same data space to provide the unrelated experience (C1 in Table 1). They match the relevant experience in the number of queries and in the interaction needed, but do not involve the tested knowledge.

**Expert annotation.** Eight experts review the queries, the data each reference program reads, and the rubrics for naturalness and correctness (Appendix J). All queries also pass an automated release gate that checks for answer leakage, grading integrity, and reproducibility (Appendix I).

Table 3: Main results on DataIntern-Lite. **Top-1**, Top-2, Top-3 per column.

| | Procedural Reuse | | | Data Space Understanding | | | | Data Space Changes | | | | Overall | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Model** | Acc@1 ↑ | Acc@3 ↑ | Tool@3 ↓ | Cold ↑ | Use ↑ | LG ↑ | Gap ↑ | New ↑ | Recur ↓ | Retain ↑ | Joint ↑ | Acc. w/ exp. ↑ | $/solve ↓ |
| GPT-6-Astra | **1.00** | **1.00** | 0.22 | 0.44 | 0.78 | **0.28** | 0.61 | 0.63 | 0.13 | 0.63 | 0.50 | 0.80 | 0.93 |
| GPT-6-Sol | **1.00** | **1.00** | 0.28 | 0.44 | 0.68 | 0.22 | 0.46 | 0.63 | 0.25 | 0.63 | 0.38 | 0.77 | 0.17 |
| GPT-6-Luna | **1.00** | **1.00** | 0.31 | 0.50 | 0.72 | 0.26 | 0.67 | 0.44 | 0.31 | 0.75 | 0.25 | 0.72 | 0.023 |
| GPT-5.6-Sol | 0.96 | 0.96 | 0.25 | 0.43 | 0.65 | 0.22 | 0.55 | 0.63 | **0.00** | 0.38 | 0.38 | 0.75 | 0.41 |
| DeepSeek-V4-Pro | 0.96 | 0.89 | 0.14 | 0.54 | 0.80 | 0.19 | **0.77** | 0.63 | 0.25 | 0.75 | 0.38 | 0.77 | 0.77 |
| DeepSeek-V4.1-Flash | **1.00** | **1.00** | 0.14 | 0.59 | 0.82 | 0.26 | 0.70 | **0.75** | 0.13 | **1.00** | **0.75** | 0.86 | 0.013 |
| Gemini-3.1-Pro | 0.59 | 0.82 | 0.21 | 0.59 | 0.82 | 0.15 | 0.50 | 0.63 | 0.31 | 0.63 | 0.25 | 0.75 | 0.36 |
| GLM-5.3 | 0.96 | 0.82 | 0.30 | 0.54 | 0.67 | 0.17 | 0.36 | 0.50 | 0.44 | 0.88 | 0.38 | 0.66 | 0.11 |
| GLM-5.3-Flash | 0.96 | 0.89 | 0.25 | 0.41 | 0.65 | 0.24 | 0.52 | 0.57 | 0.29 | 0.57 | 0.29 | 0.70 | **0.012** |
| Grok-4.6 | 0.89 | **1.00** | **0.13** | **0.80** | **0.93** | 0.20 | 0.73 | **0.75** | 0.13 | **1.00** | **0.75** | **0.89** | 0.36 |
| HY4-Preview | 0.96 | 0.96 | 0.15 | 0.48 | 0.70 | 0.20 | 0.41 | 0.50 | 0.25 | 0.75 | 0.50 | 0.72 | 0.043 |
| Kimi-K3 | 0.74 | 0.93 | 0.21 | 0.44 | 0.76 | 0.26 | 0.70 | 0.63 | 0.13 | 0.63 | 0.50 | 0.77 | 0.19 |
| Qwen-3.7-Max | 0.96 | 0.96 | 0.26 | 0.41 | 0.57 | −0.04 | −0.15 | 0.50 | 0.25 | 0.63 | 0.38 | 0.68 | 0.14 |

**Data statistics.** Table 2 summarizes DataIntern, and Appendix F gives the full statistics. We release two tiers. DataIntern-Full contains all 742 queries, and DataIntern-Lite is a reviewed subset of 193 queries whose tests leave clear room to learn.

## 4 Experiments

We organize the evaluation around four Research Questions. **RQ1 (procedural reuse):** Can data agents reuse procedures to reduce cost without losing accuracy? **RQ2 (data space understanding):** Does understanding the data space help data agents answer queries that need data space knowledge? **RQ3 (data space changes):** When conditions change, can data agents update their experience and still use the old knowledge where it applies? **RQ4 (system design):** How do the model, the harness, and the memory mechanism together shape the learning capabilities of data agents?

### 4.1 Setup

**Memory mechanisms.** We use the settings of Table 1. Between sessions, experience is carried by one of four memory mechanisms. **Session logs** copy earlier sessions into the target prompt. The short version keeps only the last message of each session, and the full version keeps every command and its output. **Structured memory** stores rules plus executable programs written by the data agent. **Mem0** (Chhikara et al., 2025) stores facts in a vector store and retrieves them by query. **Zep** (Rasmussen et al., 2025) keeps a temporal knowledge graph whose facts carry validity intervals.

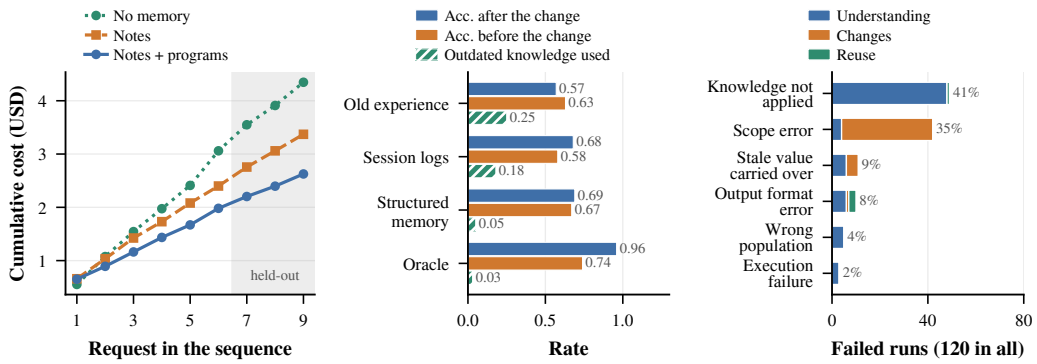
**Models, harnesses, and tiers.** The reference configuration is GPT-5.6-Sol on the Codex harness with structured memory. We evaluate thirteen frontier LLMs (Table 24) and six agent harnesses: four general-purpose coding agents, Spider-Agent (Lei et al., 2025) for enterprise data agent tasks, and a ReAct-based data agent (Table 5a; Appendix C). Model comparisons use DataIntern-Lite (193 queries, 49 tests), harness comparisons use DataIntern-Lite with three models, and the memory and cost analyses use DataIntern-Full (742 queries, 126 tests) on the reference model.

**Metrics and statistics.** We use the metrics of §2. In the main table, data space changes use the setting where the data agent updates its memory (B2), and *Acc. w/ exp.* is the mean of Acc@3, Use, and New. Seeds and statistical tests are described in Appendix B.

### 4.2 Main Results

Table 3 compares all thirteen models. Grok-4.6 and DeepSeek-V4.1-Flash reach the highest accuracy with experience, and GPT-6-Astra gains the most from it. DeepSeek-V4.1-Flash reaches its accuracy of 0.86 at $0.013 per solved query, about 70 times less than GPT-6-Astra.

**Procedural reuse makes data agents much cheaper.** By the third run, every model needs at most a third of the tool calls of the first run, and accuracy holds or rises for most models. Models that often miss the first run tend to catch up on later runs, as Gemini-3.1-Pro rises from 0.59 to 0.82 and Kimi-K3 from 0.74 to 0.93.




(a) Learning curve. (b) Data space changes. (c) Causes of failure.

Figure 3: Learning curve, data space changes, and causes of failure (GPT-5.6-Sol, Codex, Full).

**Relevant experience helps almost every model.** All models except Qwen-3.7-Max answer more understanding queries with relevant than with unrelated experience, with gains of 15 to 28 points. The size of the gain does not follow how well a model solves queries on its own. GPT-6-Astra and Qwen-3.7-Max start from similar accuracy without experience (0.44 and 0.41), yet GPT-6-Astra gains 28 points and Qwen-3.7-Max loses 4.

**Data space changes remain the hardest scenario.** Only Grok-4.6 and DeepSeek-V4.1-Flash answer correctly before and after a change in most tests (Joint 0.75), and every other model reaches at most 0.50. These models usually keep one side right at the cost of the other. GPT-5.6-Sol never repeats the outdated value (Recur 0.00) but answers only 0.38 of the queries before the change, while GLM-5.3 answers 0.88 of them but repeats the outdated value in 0.44 of the answers after it.

### 4.3 RQ1: Procedural Reuse

**One earlier run is enough to cut most of the cost.** On the reuse chains of DataIntern-Full, the second run needs about a quarter of the tool calls of the first run and half of its cost, and accuracy rises slightly (Table 9 in Appendix A.4). The data agent reuses the procedure it has written instead of exploring the data space again. When the same request repeats many times, reusable programs lower the average cost per request by about 40 % compared with running each request from scratch (Figure 3a), and data agents build such programs mainly when asked.

**Reuse cannot fix a wrong procedure.** Chains that start with a correct answer almost always stay correct, while most chains that start with a wrong answer stay wrong. The data agent copies the first procedure together with its mistakes and does not check it again. Reuse makes data agents faster, but it does not make them more reliable.

**Finding 1.** Reuse makes data agents much cheaper but not more reliable. An early mistake repeats in later runs, and data agents need to check the procedures they reuse.

### 4.4 RQ2: Data Space Understanding

**The gain comes from data space knowledge.** Table 4 compares all settings on GPT-5.6-Sol. Unrelated experience from the same data space does not help, while experience that contains the data space knowledge raises accuracy with every memory mechanism. Removing only this knowledge from structured memory halves the gain. The gain is largest on the queries that data agents cannot solve on their own (Table 5c), and relevant experience also halves the cost of answering a query (Table 5b). The oracle shows that there is still room to gain.

**Recording the knowledge in a usable form matters more than keeping a longer history.** Structured memory, Mem0, and Zep record the knowledge itself and reach the highest accuracy. Short session logs keep only the final message and often miss the knowledge. Full logs contain the knowledge together with many commands and outputs, and help less than structured memory and Mem0. A data agent learns from experience only when the knowledge is written down in a usable form.

Table 4: Memory mechanisms on the reference model. Seeds are given in Appendix A.12. /: not applicable. Full logs are compared with unrelated full logs. **Top-1**, Top-2, Top-3 per column.

| | Data Space Understanding | | | | | Data Space Changes | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Setting** | Use ↑ | LG ↑ | Gap ↑ | Neg. ↓ | $/solve ↓ | New ↑ | Recur ↓ | Retain ↑ | $/solve ↓ | Exp. tokens |
| No (C0) or old (B0) experience | 0.60 | / | / | / | 1.14 | 0.57 | 0.25 | 0.63 | 1.24 | 0 |
| Unrelated experience (C1) | 0.58 | 0.00 | 0.00 | / | 1.25 | / | / | / | / | 105 |
| Session logs | 0.63 | 0.05 | 0.19 | 0.16 | 0.97 | 0.68 | 0.18 | 0.58 | 0.95 | 87 |
| Structured memory | 0.70 | **0.13** | **0.46** | 0.10 | **0.49** | **0.69** | 0.05 | **0.67** | **0.38** | 325 |
| Structured memory, knowledge K removed | 0.65 | 0.07 | 0.27 | 0.13 | 0.57 | / | / | / | / | 330 |
| Session logs, full log | 0.67 | 0.10 | 0.35 | 0.11 | 0.60 | 0.53 | **0.03** | 0.63 | 0.88 | 17242 |
| Mem0 | **0.71** | **0.13** | **0.46** | **0.08** | 0.87 | 0.53 | 0.21 | 0.63 | 1.24 | 173 |
| Zep | 0.68 | 0.11 | 0.40 | **0.08** | 0.82 | 0.55 | 0.24 | 0.53 | 1.30 | 204 |
| Oracle (C3 / B3) | 0.85 | 0.27 | 1.00 | 0.02 | 0.54 | 0.96 | 0.03 | 0.74 | 0.43 | 148 |

Table 5: Results averaged over models on DataIntern-Lite. Cost is relative to Codex in (a) and to no experience in (b). **Top-1** per column in (a). Full results are in Appendix A.

(a) Effect of the harness.

| **Harness** | Cold ↑ | LG ↑ | Cost ↓ |
|---|---|---|---|
| Codex | 0.48 | 0.25 | 1.00 |
| Pi | 0.54 | 0.20 | **0.65** |
| Hermes Agent | **0.64** | 0.19 | 1.59 |
| LangChain-DeepAgent | 0.34 | 0.22 | 1.90 |
| Spider-Agent | 0.17 | 0.08 | 2.99 |
| ReAct-based data agent | 0.30 | **0.26** | 1.77 |

(b) Effect of experience.

| **Metric** | No exp. | Relevant |
|---|---|---|
| Accuracy | 0.48 | 0.75 |
| Tool calls | 19 | 8 |
| Input tokens (k) | 432 | 148 |
| Output tokens (k) | 8.0 | 3.9 |
| Cost per query | 1.00 | 0.45 |
| Cost per solve | 1.00 | 0.28 |

(c) Effect of difficulty.

| **Setting** | Hard | Medium | Easy |
|---|---|---|---|
| No experience | 0.12 | 0.52 | 0.85 |
| Unrelated | 0.17 | 0.54 | 0.85 |
| Relevant | 0.53 | 0.82 | 0.94 |
| Oracle | 0.93 | 0.89 | 0.94 |
| **Gain (LG)** | **+0.37** | **+0.28** | **+0.09** |
| Targets | 17 | 12 | 15 |

**Finding 2.** Learning a data space depends on what a data agent chooses to remember. Recording the knowledge itself helps more than keeping the full history, and future data agents need to learn what to write down.

### 4.5 RQ3: Data Space Changes

**Data agents learn the new knowledge but do not know where it applies.** On GPT-5.6-Sol, about one in four answers after the change still uses the outdated knowledge when the data agent keeps its old experience (Figure 3b). Updating structured memory makes this rare. On the query where the old knowledge still holds, every memory mechanism stays below the oracle (0.53–0.67 against 0.74; Table 4). The data agent records the new knowledge but does not track which queries each version covers. Across models the effect of an update varies, and several models reuse outdated knowledge more often after updating their memory (Appendix A.2).

**Finding 3.** Data agents fail mainly by ignoring recorded knowledge (41 % of failures) or by applying it where it does not hold (35 %), even with memory that tracks when each fact is valid. The next step is memory that records which version of the knowledge applies to each query.

### 4.6 RQ4: Model, Harness, and Memory

**Experience closes the gap between cheap and strong models.** With structured memory, GPT-6-Luna and DeepSeek-V4.1-Flash answer 72 % and 82 % of the understanding queries, close to GPT-6-Astra (78 %), at a small fraction of its cost (Table 3). Without experience, all three models solve fewer than 60 % of these queries. On a data space it has worked in, a low-cost model with the right experience comes close to a much stronger one.

**Harnesses differ in accuracy without experience, but the learning gain stays similar.** With structured memory fixed (Table 5a), accuracy without experience varies widely across harnesses, while the learning gain varies much less and does not follow cold accuracy. The harness shapes how a data agent explores the data space, but what it learns depends more on the experience it carries.

**All memory mechanisms help on understanding, but only some handle data space changes well.** Structured memory, Mem0, and Zep reach similar accuracy on understanding, and structured memory is the cheapest (Table 4). On data space changes, Zep reuses outdated knowledge more often than the other memory mechanisms despite its validity intervals, while structured memory and full logs reuse it least.

Table 6: Comparison with benchmarks. Matched control: same-size experience without knowledge. Changes: scored before and after. Cost: building and using experience. ✓ yes, △ partial, × no.

| Benchmark | Field | Tasks | Resources | Data agent tasks | Shared knowl. | Matched control | Changes | Cost |
|---|---|---|---|---|---|---|---|---|
| *Data agent benchmarks: each query is solved from a fresh state* | | | | | | | | |
| Spider 2.0 (Lei et al., 2025) | Text-to-SQL | 632 | 213 DBs | ✓ | × | × | × | × |
| BIRD-Interact (Huo et al., 2026) | Interactive text-to-SQL | 600 | 22 DBs | ✓ | × | × | × | △ |
| KramaBench (Lai et al., 2026) | Data science pipelines | 104 | 24 sources, 1.7k files | ✓ | × | × | × | × |
| DAComp (Lei et al., 2026) | Data eng. and analysis | 210 | 173 schemas | ✓ | × | × | × | × |
| FDABench (Wang et al., 2026b) | Data analysis | 2,007 | 139 DBs | ✓ | × | × | × | △ |
| *Agent learning benchmarks: experience carries across tasks* | | | | | | | | |
| LifelongAgentBench (Zheng et al., 2025) | Lifelong learning | 1,396 | 3 envs (DB, OS, KG) | △ | ✓ | △ | × | × |
| CL-Bench (Asawa et al., 2026) | Continual learning | 301 | 6 envs, 1 DB | △ | ✓ | △ | ✓ | ✓ |
| PATH-Bench (Yang et al., 2026) | Learning from history | 240 | 2 suites, no data | × | ✓ | ✓ | △ | × |
| Crystallization (Wang et al., 2026a) | Experience reuse | 1,534 | 11 DBs, tables only | △ | ✓ | ✓ | × | ✓ |
| **DataIntern** | **Data agent learning capability** | **742** | **15 spaces, 450 tables, 13.6k docs, 68 GB** | ✓ | ✓ | ✓ | ✓ | ✓ |

### 4.7 Error Analysis

**Most failures come from misusing or ignoring recorded knowledge.** A labeling model assigns a cause to each failed run of GPT-5.6-Sol with experience, and 33 of 35 labels checked by hand agree (Figure 3c; Appendix A.6). Two causes explain three quarters of the failures. A *scope error* applies knowledge outside the cases it covers, such as the old knowledge after a change, and explains most failures on data space change tests (a case in Appendix A.1). It remains the main cause even when the oracle states the new knowledge. *Knowledge not applied* computes the answer as if the knowledge did not exist. It is most common on understanding tests, and with session logs it often means that the knowledge was never recorded.

## 5 Related Work

**Data agent benchmarks.** Early text-to-SQL benchmarks typically cover single databases (Yu et al., 2018; Li et al., 2023). Spider 2.0 (Lei et al., 2025) extends this to enterprise-level databases with file systems and multi-step tool use. BIRD-Interact (Huo et al., 2026) adds clarification and follow-up queries within one session but does not carry experience across sessions. Recent data agent benchmarks cover more types of data workloads. DAComp (Lei et al., 2026) covers multi-step data engineering and analysis pipelines over real repositories, KramaBench (Lai et al., 2026) evaluates data science pipelines over mixed sources including files and databases, and FDABench (Wang et al., 2026b) evaluates analytical queries over heterogeneous data. Other benchmarks target long analyses (Xu et al., 2026), workspace file dependencies (Tang et al., 2026), and verifiable analytics (Hu et al., 2024; Li et al., 2026). All of these benchmarks evaluate each query from a fresh state and do not test whether a data agent learns from its own experience.

**Continual learning benchmarks.** LifelongAgentBench (Zheng et al., 2025) carries experience across agent tasks such as running terminal commands or querying a knowledge graph, but none of them involves enterprise data analytics. CL-Bench (Asawa et al., 2026) tests whether general agents adapt to changes in six stateful environments. It is not designed for enterprise data agent tasks, and its only data environment is a single database without files or documents. PATH-Bench (Yang et al., 2026) compares helpful and misleading earlier sessions for code and tool-use agents, without enterprise data analytics. Crystallization (Wang et al., 2026a) separates reusing earlier answers from transferring knowledge to new questions in text-to-SQL, but uses tables only, without data space changes. DataIntern alone covers data agent tasks, data space knowledge shared across queries, matched controls, data space changes, and learning cost over heterogeneous data spaces (Table 6).

## 6 Conclusion

In this paper, we introduce DataIntern, a benchmark for evaluating the learning capabilities of data agents in three scenarios, *procedural reuse*, *data space understanding*, and *data space changes*. Current data agents reuse procedures well and gain from relevant experience, but they often apply knowledge where it does not hold. Promising directions for future data agents include deciding which knowledge to use under which conditions, storing it in a usable form, and balancing cost and quality as they keep learning over an enterprise data space.

## AI USE STATEMENT

In this work, we used generative AI tools to collect, clean, and generate the benchmark data (§3), to support data analysis, and to polish the writing. We have reviewed all AI-assisted work: every released query is verified by execution and reviewed by experts. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

## ETHICS STATEMENT

All data spaces are built from public data. The construction process reviews the license of every source and removes or replaces sensitive identifiers before release. The enterprise feedback used to design the data spaces is anonymized before release. The release documents data provenance, access restrictions, and any limitations on redistribution.

## REPRODUCIBILITY STATEMENT

The release includes the queries, the data space builders and snapshots where licensing permits, the reference programs, the grading items and rubrics, the knowledge unit and test annotations, the workload orders, and the scripts that reproduce every result under the evaluation settings of §2. Model versions, harness versions, list prices, and seeds are given in Appendix B.

## REFERENCES

Shubham Agarwal, Asim Biswal, Sepanta Zeighami, Alvin Cheung, Joseph Gonzalez, and Aditya G. Parameswaran. Arming data agents with tribal knowledge. *arXiv preprint arXiv:2602.13521*, 2026.

Parth Asawa, Christopher M. Glaze, Gabriel Orlanski, Ramya Ramakrishnan, Benji Xu, Asim Biswal, Vincent Sunn Chen, Frederic Sala, Matei Zaharia, and Joseph E. Gonzalez. Continual learning bench: Evaluating frontier AI systems in real-world stateful environments. *arXiv preprint arXiv:2606.05661*, 2026.

Asim Biswal, Chuan Lei, Xiao Qin, Aodong Li, Balakrishnan Narayanaswamy, and Tim Kraska. AgentSM: Semantic memory for agentic text-to-SQL. *arXiv preprint arXiv:2601.15709*, 2026.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. In *Proceedings of the 28th European Conference on Artificial Intelligence (ECAI)*, pp. 2993–3000, 2025. doi: 10.3233/FAIA251160.

Databricks. Genie. `https://www.databricks.com/product/business-intelligence/genie`, 2025.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. *arXiv preprint arXiv:2609.19969*, 2026a.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026b.

Earendil Works. Pi. `https://github.com/earendil-works/pi`, 2025.

Jiajie Fu, Junwen Chen, Mengzhao Wang, Aoxiang He, Maojia Sheng, Xiangyu Ke, Yifan Zhu, and Yunjun Gao. VikingMem: A memory base management system for stateful LLM-based applications. *Proceedings of the VLDB Endowment*, 19(12):4344–4357, 2026.

GLM-5-Team. GLM-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Google DeepMind. Gemini 3.1 Pro model card. `https://deepmind.google/models/model-cards/gemini-3-1-pro/`, 2026.

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

Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, et al. InfiAgent-DABench: Evaluating agents on data analysis tasks. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, and Kang Liu. DA-Code: Agent data science code generation benchmark for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024.

Nan Huo, Xiaohan Xu, Jinyang Li, Per Jacobsson, Shipei Lin, Bowen Qin, Binyuan Hui, Xiaolong Li, Ge Qu, Shuzheng Si, Linheng Han, Edward Alexander, Xintong Zhu, Rui Qin, Ruihan Yu, Yiyao Jin, Feige Zhou, Weihao Zhong, Yun Chen, Hongyu Liu, Chenhao Ma, Fatma Ozcan, Yannis Papakonstantinou, and Reynold Cheng. BIRD-Interact: Re-imagining text-to-SQL evaluation via lens of dynamic interactions. In *The Fourteenth International Conference on Learning Representations*, 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 *The Thirteenth International Conference on Learning Representations*, 2025.

Kimi Team. Kimi K3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Eugenie Lai, Gerardo Vitagliano, Ziyu Zhang, Om Chabra, Sivaprasad Sudhir, Anna Zeng, Anton A. Zabreyko, Chenning Li, Ferdi Kossmann, Jialin Ding, et al. KramaBench: A benchmark for AI systems on data-to-insight pipelines over data lakes. In *The Fourteenth International Conference on Learning Representations*, 2026.

LangChain. Deep Agents. `https://github.com/langchain-ai/deepagents`, 2025.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, et al. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *The Thirteenth International Conference on Learning Representations*, 2025.

Fangyu Lei, Jinxiang Meng, Yiming Huang, Junjie Zhao, Yitong Zhang, Jianwen Luo, Xin Zou, Ruiyi Yang, Wenbo Shi, Yan Gao, et al. DAComp: Benchmarking data agents across the full data intelligence lifecycle. In *The Fourteenth International Conference on Learning Representations*, 2026.

Boyan Li, Zhuowen Liang, Yupeng Xie, Xiaotian Lin, Tianqi Luo, Xinyu Liu, Yizhang Zhu, Zhangyang Peng, Yuan Li, Zhengxuan Zhang, et al. DataSpace: Benchmarking data agents for verifiable analytics over heterogeneous workspaces. *arXiv preprint arXiv:2608.03451*, 2026.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can LLM already serve as a database interface? A BIg bench for large-scale database grounded text-to-SQLs. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Bhavana Dalvi Mishra, Abhijeetsingh Meena, Aryan Prakhar, Tirth Vora, Tushar Khot, Ashish Sabharwal, and Peter Clark. DiscoveryBench: Towards data-driven discovery with large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Nous Research. Hermes Agent. `https://github.com/NousResearch/hermes-agent`, 2025.

OpenAI. Codex CLI. `https://github.com/openai/codex`, 2025.

OpenAI. GPT-5.6 system card. https://deploymentsafety.openai.com/gpt-5-6, 2026a.

OpenAI. GPT-6 Astra system card. https://deploymentsafety.openai.com/gpt-6-astra, 2026b.

Charles Packer, Vivian Fang, Shishir G. Patil, Kevin Lin, Sarah Wooders, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *CoRR*, abs/2310.08560, 2023. doi: 10.48550/ARXIV.2310.08560.

Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023.

Qwen Team. Qwen3.7: The agent frontier. https://www.alibabacloud.com/blog/qwen3-7-the-agent-frontier_603154, 2026.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

Snowflake. Cortex analyst. https://docs.snowflake.com/en/user-guide/snowflake-cortex/cortex-analyst, 2025.

Zirui Tang, Xuanhe Zhou, Yumou Liu, Linchun Li, Yukai Wu, Weizheng Wang, Hongzhang Huang, Wei Zhou, Jun Zhou, Jiachen Song, et al. Workspace-bench 1.0: Benchmarking AI agents on workspace tasks with large-scale file dependencies. *arXiv preprint arXiv:2605.03596*, 2026.

Tencent Hy Team. Hy4 preview model card. https://huggingface.co/tencent/Hy4-preview, 2026.

Kapil Vaidya, Jialin Ding, Sebastian Kosak, David Kernert, Chuan Lei, Xiao Qin, Abhinav Tripathy, Ramesh Balan, Balakrishnan Narayanaswamy, and Tim Kraska. TailorSQL: An NL2SQL system tailored to your query workload. *arXiv preprint arXiv:2505.23039*, 2025.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024.

Jiaqian Wang, Yutao Qi, Wenjin Hou, Yuanxi Che, and Muning Wen. From test-time scaling to reusable memory: Measuring crystallization in text-to-SQL. *arXiv preprint arXiv:2608.07213*, 2026a.

Ziting Wang, Shize Zhang, Haitao Yuan, Jinwei Zhu, Wei Dong, and Gao Cong. FDABench: A benchmark for data agents on analytical queries over heterogeneous data. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 9985–9996. ACM, 2026b. doi: 10.1145/3770855.3817454.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

xAI. Grok 4.6 model card. https://media.x.ai/v1/website/card-4p6-4cd2dc57.pdf, 2026.

Kewei Xu, Xiaoben Lu, Shuofei Qiao, Zihan Ding, Haoming Xu, Lei Liang, and Ningyu Zhang. LongDS-Bench: On the failure of long-horizon agentic data analysis. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing*, 2026.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, 2025.

Xidong Yang, Xingyi Zhang, Wenhao Li, Wenyan Liu, Junjie Sheng, Yun Hua, Wei Yin, Tao Fang, Chuyun Shen, and Xiangfeng Wang. PATH-Bench: Path-dependent evaluation of lifelong agents. *arXiv preprint arXiv:2608.01149*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. 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*, 2018.

Z.ai. GLM-5.3-Flash model card. `https://huggingface.co/zai-org/GLM-5.3-Flash`, 2026.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, ZhongZhi Li, Yingying Zhang, Le Song, and Qianli Ma. LifelongAgentBench: Evaluating LLM agents as lifelong learners. *arXiv preprint arXiv:2505.11942*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, et al. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*, 2023.

# A Additional Results

This section expands the results of §4. Unless stated otherwise, model comparisons use DataIntern-Lite with Codex and structured memory, and analyses of the reference model use DataIntern-Full.

## A.1 Case study: a scope error in the banking data space

This case comes from the data space change test on the Regulation C asset-size exemption (§3.2). Small institutions do not have to report their mortgage lending when their assets fall below a threshold. The threshold is $47 million for 2020, measured on the report of 2019-12-31, and $58 million for 2025, measured on the report of 2024-12-31. Both Federal Register notices are in the data space.

GPT-5.6-Sol with structured memory solves the change-point query, which asks for the rule that governs the 2025 review. It finds the 2025 notice and writes it into its notes: the notice number, the new threshold of $58 million, and the fact that institutions at or below it are exempt from 2025 reporting. It also stores the query it ran, which selects the institutions on the 2024-12-31 report with assets of at most $58 million and returns 315 institutions.

The retention query then asks to rebuild the exemption exhibit used at a March 2020 planning meeting. The data agent runs the stored query unchanged and reports 315 institutions, the largest with $57.98 million in assets. The correct answer uses the $47 million threshold on the 2019-12-31 report and gives 411 institutions, the largest with $46.89 million. The notes say that the new threshold applies to 2025, but the stored query carries no such condition, and the data agent does not check which rule governs the requested year. The knowledge is recorded correctly, and the failure comes from applying it where it does not hold.

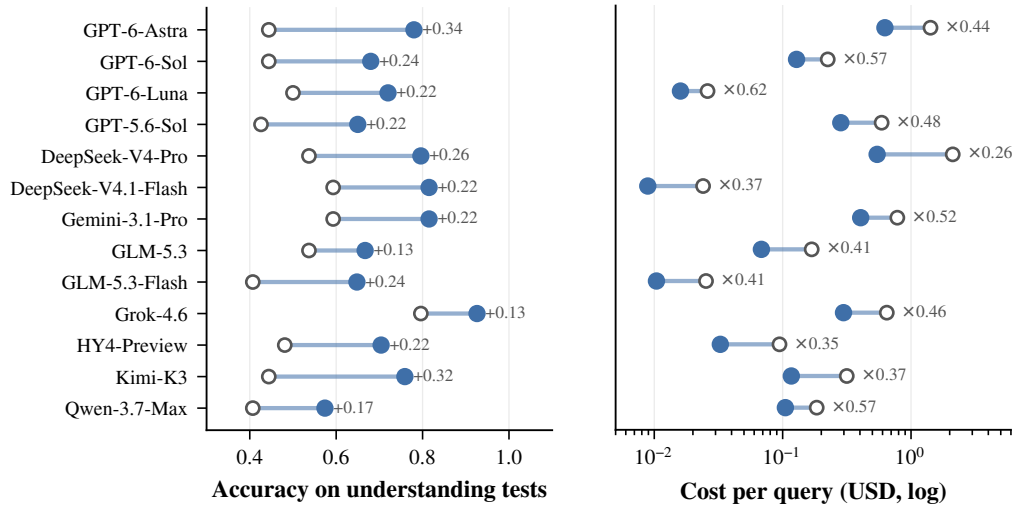
## A.2 Results per model

**Every model gains from relevant experience except Qwen-3.7-Max, and every model is at least as accurate with the oracle.** Table 7 gives the four understanding settings for all thirteen models, and Figure 4 places them against their cost.

Table 7: Data space understanding per model (Lite, Codex). **Top-1**, Top-2, Top-3 per column.

| | Accuracy by setting | | | | Learning | | |
|---|---|---|---|---|---|---|---|
| **Model** | Cold (C0) | Unrelated (C1) | Relevant (C2) | Oracle (C3) | LG ↑ | Gap ↑ | Neg. ↓ |
| GPT-6-Astra | 0.44 | 0.50 | 0.78 | 0.96 | **0.28** | 0.61 | 0.04 |
| GPT-6-Sol | 0.44 | 0.46 | 0.68 | 0.94 | 0.22 | 0.46 | **0.00** |
| GPT-6-Luna | 0.50 | 0.46 | 0.72 | 0.85 | 0.26 | 0.67 | **0.00** |
| GPT-5.6-Sol | 0.43 | 0.43 | 0.65 | 0.83 | 0.22 | 0.55 | 0.04 |
| DeepSeek-V4-Pro | 0.54 | 0.61 | 0.80 | 0.85 | 0.19 | **0.77** | 0.04 |
| DeepSeek-V4.1-Flash | 0.59 | 0.56 | 0.82 | 0.93 | 0.26 | 0.70 | 0.04 |
| Gemini-3.1-Pro | 0.59 | 0.67 | 0.82 | 0.96 | 0.15 | 0.50 | 0.04 |
| GLM-5.3 | 0.54 | 0.50 | 0.67 | 0.96 | 0.17 | 0.36 | 0.15 |
| GLM-5.3-Flash | 0.41 | 0.41 | 0.65 | 0.87 | 0.24 | 0.52 | 0.04 |
| Grok-4.6 | **0.80** | **0.72** | **0.93** | **1.00** | 0.20 | 0.73 | **0.00** |
| HY4-Preview | 0.48 | 0.50 | 0.70 | **1.00** | 0.20 | 0.41 | 0.07 |
| Kimi-K3 | 0.44 | 0.50 | 0.76 | 0.87 | 0.26 | 0.70 | 0.04 |
| Qwen-3.7-Max | 0.41 | 0.61 | 0.57 | 0.85 | −0.04 | −0.15 | 0.19 |

**On average, only the oracle keeps both the old and the new knowledge right.** Table 8 gives the four settings of the data space change tests per model. Averaged over the models, accuracy on the query where the old knowledge still holds is 0.71–0.73 in every setting that keeps experience and 0.93 with the oracle. The outdated value appears in 18–22 % of the answers after the change, and in 1 % with the oracle. Per model, the effect of updating structured memory (B2 against B0) on retention varies: some models retain less, some retain more.




(a) Accuracy. (b) Cost per query.

Figure 4: Data space understanding per model (Lite, Codex). Hollow: no experience; filled: with experience.

Table 8: Data space changes per model (Lite, Codex). **Top-1**, Top-2, Top-3 per column.

| | Old experience (B0) | | | Session logs (B1) | | | Structured memory (B2) | | | Oracle (B3) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Model** | New | Recur | Retain | New | Recur | Retain | New | Recur | Retain | New | Recur | Retain |
| GPT-6-Astra | 0.56 | 0.19 | 0.50 | 0.69 | 0.19 | 0.75 | 0.63 | 0.13 | 0.63 | **1.00** | **0.00** | **1.00** |
| GPT-6-Sol | 0.69 | 0.13 | 0.75 | 0.69 | 0.19 | 0.75 | 0.63 | 0.25 | 0.63 | **1.00** | **0.00** | 0.88 |
| GPT-6-Luna | 0.50 | 0.38 | 0.88 | 0.38 | 0.31 | 0.50 | 0.44 | 0.31 | 0.75 | **1.00** | **0.00** | 0.75 |
| GPT-5.6-Sol | 0.63 | 0.13 | 0.50 | 0.63 | 0.19 | 0.38 | 0.63 | **0.00** | 0.38 | **1.00** | **0.00** | 0.88 |
| DeepSeek-V4-Pro | 0.63 | 0.25 | 0.75 | 0.56 | 0.31 | **1.00** | 0.63 | 0.25 | 0.75 | **1.00** | **0.00** | **1.00** |
| DeepSeek-V4.1-Flash | **0.75** | 0.13 | 0.75 | **0.81** | **0.06** | 0.88 | **0.75** | 0.13 | **1.00** | **1.00** | **0.00** | **1.00** |
| Gemini-3.1-Pro | 0.63 | 0.19 | 0.88 | 0.75 | 0.13 | 0.75 | 0.63 | 0.31 | 0.63 | 0.94 | 0.06 | **1.00** |
| GLM-5.3 | 0.50 | 0.38 | 0.75 | 0.63 | 0.25 | 0.50 | 0.50 | 0.44 | 0.88 | 0.94 | **0.00** | **1.00** |
| GLM-5.3-Flash | **0.75** | **0.00** | 0.63 | **0.81** | **0.06** | 0.38 | 0.57 | 0.29 | 0.57 | 0.88 | 0.06 | **1.00** |
| Grok-4.6 | **0.75** | 0.13 | **1.00** | 0.75 | 0.19 | **1.00** | **0.75** | 0.13 | **1.00** | **1.00** | **0.00** | **1.00** |
| HY4-Preview | **0.75** | 0.13 | 0.75 | 0.75 | 0.13 | 0.88 | 0.50 | 0.25 | 0.75 | **1.00** | **0.00** | **1.00** |
| Kimi-K3 | 0.56 | 0.25 | 0.75 | 0.63 | 0.25 | 0.75 | 0.63 | 0.13 | 0.63 | 0.94 | **0.00** | 0.88 |
| Qwen-3.7-Max | 0.38 | 0.38 | 0.63 | 0.56 | 0.13 | 0.75 | 0.50 | 0.25 | 0.63 | **1.00** | **0.00** | 0.75 |

### A.3 Where experience helps

**Experience helps most on the queries that data agents cannot solve on their own.** Figure 5 groups the understanding targets by the share of the thirteen models that solve them without experience. In the hard group, relevant experience raises accuracy from 0.17 (unrelated experience) to 0.53; in the easy group, from 0.85 to 0.94. Unrelated experience helps little in every group.

### A.4 Reuse chains

**The saving arrives on the second run.** Table 9 gives the chain results on DataIntern-Full. Token ratios fall less than tool calls because the resumed session re-reads its earlier context.

### A.5 Harnesses

**The harness mainly changes cold-start accuracy.** Table 10 crosses three models with six harnesses. Cold-start accuracy and the learning gain show no clear relation ($r = 0.20$, Figure 6).

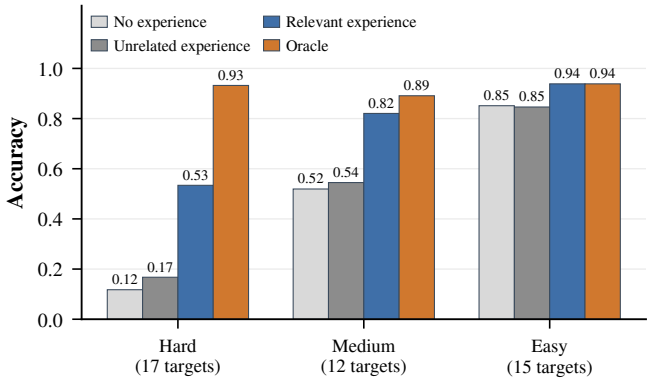



Figure 5: Data space understanding by difficulty (Lite, Codex). Hard: fewer than a third of models solve cold; easy: more than two thirds.

Table 9: Short chains (GPT-5.6-Sol, Codex, Full, three seeds). Ratios are medians relative to the first run.

| Run | $n$ | Acc | Tool calls | Tool ratio | Token ratio | Wall ratio | $ ratio (cached) | $ ratio (no cache) |
|---|---|---|---|---|---|---|---|---|
| 1 | 111 | 0.387 | 21 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 2 | 111 | 0.432 | 5 | 0.25 | 0.52 | 0.47 | 0.51 | 0.51 |
| 3 | 111 | 0.477 | 3 | 0.19 | 0.48 | 0.37 | 0.41 | 0.47 |

### A.6 Failure causes per setting

**With session logs, failures come from knowledge that was never recorded; after a change, they come from rules applied in the wrong place.** The labeled runs are the later runs of reuse chains whose first run is correct, the understanding targets with session logs or structured memory, and the queries before and after each change. A labeling model reads each failed run together with its failing checks (expected and returned values) and its command trace, and assigns one of six causes; on a stratified sample of 35 runs checked by hand, 33 labels agree. Table 11 splits the 120 failures of §4.7 by experience setting. On understanding tests with session logs, 31 of 41 failures recompute with the naive definition. On data space change tests, 35 of 39 failures with carried state are scope errors, with session logs and structured memory alike.

### A.7 Long sequences

**The cost saving comes from the programs.** Table 12 and Figure 7 give the long sequences. Removing the programs brings held-out cost back to no-memory levels ($0.42 vs. $0.21).

**Notes and programs cut cost across models.** Table 13 gives the three main settings per model on the long sequences.

**Data agents build programs when asked, seldom otherwise.** Table 14 gives the six long sequences one by one. Without instruction, a program was built in one of six cases; with instruction, in all six.

### A.8 Per data space

**Experience helps most on harder data spaces.** Table 15 reports results per data space. The gain is largest in steam, aviation, IPEDS and the EU electricity market.

### A.9 Interaction and cost

**Experience cuts cost for every model.** Table 16 gives tool calls and cost per data space understanding query, averaged over queries (not tests). With structured memory, tool calls fall to 30–70 % and cost to 26–62 % for every model.

Table 10: Effect of the harness (Lite, structured memory). **Top-1** per model.

| Model | Harness | Solving | Learning | | | | Tokens (k) | | Cost (USD) | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Cold ↑ | Tool@3 ↓ | LG ↑ | Gap ↑ | Recur ↓ | 1st call input ↓ | Total /query ↓ | $/q ↓ | $/solve ↓ |
| Kimi-K3 | Codex | 0.44 | 0.21 | 0.26 | 0.70 | 0.13 | 9.5 | 142 | 0.15 | 0.19 |
| | Pi | 0.43 | 0.25 | 0.26 | 0.56 | 0.25 | 2.1 | **58** | **0.067** | **0.079** |
| | Hermes Agent | **0.65** | 0.33 | 0.20 | 0.53 | 0.25 | 11.9 | 240 | 0.22 | 0.26 |
| | LangChain-DeepAgent | 0.33 | 0.19 | **0.32** | 0.61 | 0.13 | 3.4 | 253 | 0.20 | 0.26 |
| | Spider-Agent | 0.07 | **0.00** | 0.09 | **0.72** | **0.00** | 1.8 | 150 | 0.25 | 3.92 |
| | ReAct-based data agent | 0.39 | 0.27 | 0.28 | 0.58 | 0.13 | **1.1** | 130 | 0.13 | 0.15 |
| DeepSeek V4.1-Flash | Codex | 0.59 | 0.14 | 0.26 | 0.70 | 0.13 | 10.2 | **336** | **0.011** | 0.013 |
| | Pi | **0.63** | 0.18 | 0.15 | 0.50 | 0.13 | 2.4 | 366 | **0.011** | **0.012** |
| | Hermes Agent | 0.56 | 0.28 | 0.22 | **0.71** | 0.25 | 12.0 | 366 | 0.012 | 0.014 |
| | LangChain-DeepAgent | 0.33 | **0.11** | 0.19 | 0.42 | 0.19 | 3.8 | 458 | 0.018 | 0.023 |
| | Spider-Agent | 0.17 | 0.34 | 0.06 | 0.25 | **0.00** | 1.8 | 533 | 0.053 | 0.17 |
| | ReAct-based data agent | 0.26 | 0.20 | **0.33** | 0.67 | 0.13 | **1.3** | 342 | 0.013 | 0.017 |
| GLM-5.3 Flash | Codex | 0.41 | 0.25 | **0.24** | **0.52** | 0.29 | 9.8 | 180 | 0.0096 | 0.012 |
| | Pi | 0.57 | 0.23 | 0.19 | 0.43 | 0.13 | 2.2 | **78** | **0.0053** | **0.0067** |
| | Hermes Agent | **0.70** | 0.20 | 0.13 | 0.41 | 0.13 | 12.0 | 360 | 0.022 | 0.025 |
| | LangChain-DeepAgent | 0.35 | **0.17** | 0.17 | 0.50 | 0.25 | 3.6 | 501 | 0.026 | 0.040 |
| | Spider-Agent | 0.28 | 0.18 | 0.09 | 0.19 | **0.00** | 1.7 | 230 | 0.025 | 0.065 |
| | ReAct-based data agent | 0.24 | 0.20 | 0.17 | 0.32 | 0.25 | **1.1** | 561 | 0.031 | 0.056 |

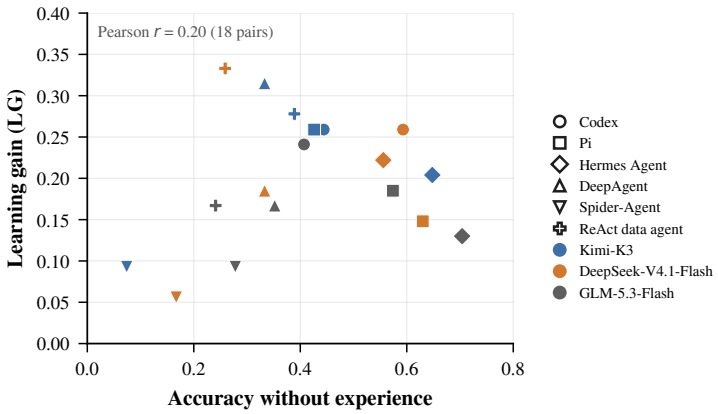



Figure 6: Cold-start accuracy vs. learning gain per model–harness pair (Lite, structured memory).

**Counting the source queries.** The cost per query above covers the target query, including reading and retrieving experience. The experience itself is built while the data agent solves the source queries, which are ordinary requests of the workload. Counting these source sessions as well, structured memory costs $0.62 per target query on the reference model (DataIntern-Full), close to $0.64 without experience. In our tests one source serves 1.75 targets on average, so the saving on the targets covers the cost of the sources. The saving per target grows with the number of later queries that use the same experience, as the long sequences show (Appendix A.7).

### A.10 Evaluation sets and knowledge in memory

**Results are stable on the common test subset.** Table 17 gives the run counts behind Table 4. Table 18 repeats the understanding columns of Table 4 on the 62 tests that every setting completed; understanding accuracy changes by at most 0.02. The B0–B3 change-test results in Table 4 use three-seed averages; other memory mechanisms and Table 18 use seed 1.

**Relevant experience helps when the knowledge reaches the memory.** For every understanding test, a judge model (Zheng et al., 2023) (GPT-6-Sol) reads the experience that the target saw and decides whether it states the knowledge K, applies it, or neither (Table 19). K reaches structured memory in 78 % of the tests, full session logs in 73 %, Mem0 in 55 %, Zep in 38 % and short session

Table 11: Failure causes by setting (GPT-5.6-Sol, Codex, Full).

| Setting | Failures | Scope error | Knowledge not applied | Stale value carried over | Wrong population | Output format error | Execution failure |
|---|---|---|---|---|---|---|---|
| Same session (reuse) | 4 | 0 | 1 | 0 | 0 | 3 | 0 |
| Session logs, understanding | 41 | 1 | 31 | 2 | 1 | 3 | 3 |
| Structured memory, understanding | 31 | 3 | 17 | 4 | 4 | 3 | 0 |
| Session logs, data space changes | 21 | 19 | 0 | 2 | 0 | 0 | 0 |
| Structured memory, data space changes | 18 | 16 | 0 | 2 | 0 | 0 | 0 |
| Oracle, data space changes | 5 | 3 | 0 | 1 | 0 | 1 | 0 |
| All | 120 | 42 (35 %) | 49 (41 %) | 11 (9 %) | 5 (4 %) | 10 (8 %) | 3 (2 %) |

Table 12: Long sequences (GPT-5.6-Sol, Codex, Full). **Top-1** per column.

| | Accuracy | | All requests | | Held-out requests | | |
|---|---|---|---|---|---|---|---|
| Setting | Observed ↑ | Held-out ↑ | Tools/req ↓ | $/req ↓ | Tools/req ↓ | $/req ↓ | Builds programs |
| No memory | **0.97** | 1.00 | 16 | 0.48 | 14 | 0.42 | / |
| Notes | **0.97** | 1.00 | 9 | 0.38 | 8.5 | 0.35 | 1/6 |
| Notes + programs | **0.97** | 1.00 | **7** | **0.29** | **6** | **0.21** | 6/6 |
| Notes, programs removed | / | 1.00 | / | / | 12.5 | 0.42 | removed |
| Asked to build programs, no memory | 0.94 | 1.00 | 16 | 0.64 | 15 | 0.50 | 0/6 |

logs in 21 %, and the unrelated experience in 2 %. Targets that saw K are solved more often than those that did not, for example 0.76 against 0.50 with structured memory. Structured memory is about three times as long as the unrelated experience. We therefore also replace only the lines that carry K with unrelated notes of the same length. K then remains in 2 % of the memories, and the gain over unrelated experience falls from 0.14 to 0.08 (Table 18), so the gain does not come from the length of the memory alone. The history behind relevant and unrelated experience takes a similar number of tool calls (median ratio 0.92–1.17).

### A.11 Robustness

**Seeds.** The reference model was run with three seeds on the cells in Table 20. Accuracy without experience, with structured memory and with the oracle varies by at most 0.05 between seeds, with unrelated experience by 0.09, and on the reuse chains by at most 0.08. The cells of the data space change tests vary by up to 0.21 because they rest on 19 tests. **Lite vs. Full.** Table 21 compares the two tiers for three models. The learning gain is positive on both tiers. See Appendix F for the definition of each tier.

**Paired tests.** Table 22 gives the paired tests behind the main claims on the reference model (procedure in Appendix B).

### A.12 Result provenance

Table 23 lists the aggregation behind each result table. Every table uses the same frozen release (742 queries, gen11-v3c). Screening runs that select DataIntern-Lite are separate from the evaluation runs: screening uses a single GPT-5.6-Sol seed, and evaluation runs use independent seeds on the released queries. The three-model Full results in Table 21 use as many tests as each model has completed (63 for GPT-5.6-Sol, 59 for the other two).

## B Experiment Details

**Models.** Table 24 lists the evaluated models, their OpenRouter identifiers, and the OpenRouter list prices used for every cost in the paper. We always use the price without temporary discounts, and compute costs from the token counts in the session records.

**Harness versions.** Codex CLI (OpenAI, 2025) 0.155.1 with reasoning effort high; Pi coding agent (Earendil Works, 2025) 0.86.0 with thinking high and its default cache warming; Hermes

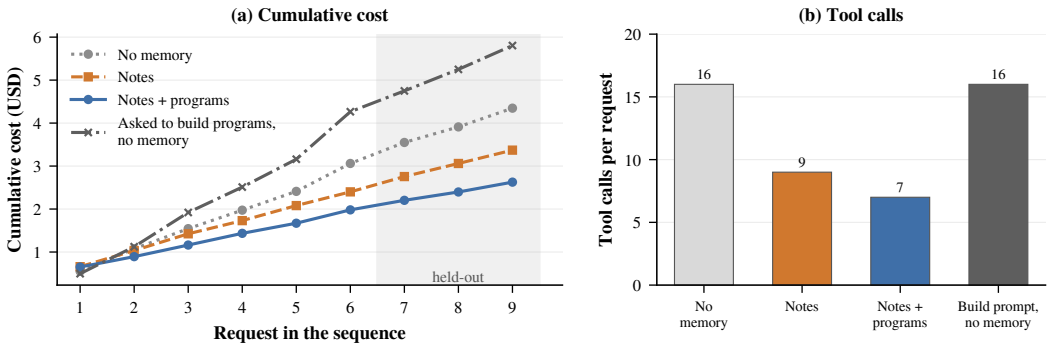



Figure 7: Long sequences (GPT-5.6-Sol, Codex, Full). Shaded: held-out requests.

Table 13: Long sequences per model. R: no memory, N: notes, S: notes + programs.

| | Held-out Acc | | | $/request | | | Builds programs | |
|---|---|---|---|---|---|---|---|---|
| **Model** | R | N | S | R | N | S | notes only | asked |
| GPT-5.6-Sol | 1.00 | 1.00 | 1.00 | 0.48 | 0.38 | 0.29 | 1/6 | 6/6 |
| DeepSeek-V4-Pro | 0.87 | 1.00 | 0.80 | 4.78 | 1.48 | 0.88 | 0/5 | 5/5 |
| DeepSeek-V4.1-Flash | 0.93 | 1.00 | 1.00 | 0.05 | 0.02 | 0.01 | 5/5 | 5/5 |
| GLM-5.3 | 1.00 | 1.00 | 1.00 | 0.15 | 0.10 | 0.07 | 0/5 | 5/5 |
| GLM-5.3-Flash | 0.75 | 1.00 | 1.00 | 0.03 | 0.01 | 0.01 | 3/6 | 6/6 |
| HY4-Preview | 0.93 | 1.00 | 1.00 | 0.13 | 0.06 | 0.06 | 5/5 | 5/5 |
| Kimi-K3 | 0.80 | 0.87 | 1.00 | 0.42 | 0.18 | 0.16 | 0/5 | 5/5 |
| Qwen-3.7-Max | 0.93 | 0.93 | 0.80 | 0.24 | 0.10 | 0.10 | 0/5 | 5/5 |
| DeepSeek-V4.1-Flash (Pi) | 0.80 | 1.00 | 1.00 | 0.05 | 0.01 | 0.01 | 5/5 | 5/5 |
| GLM-5.3-Flash (Pi) | 1.00 | 0.80 | 1.00 | 0.02 | 0.01 | 0.01 | 2/5 | 5/5 |

Agent (Nous Research, 2025) 0.14.0; LangChain-DeepAgent (LangChain, 2025) (DeepAgents 0.7.17); Spider-Agent from Spider 2.0 (spider-agent-lite, commit `cafb8673`); and the ReAct-based data agent of this paper (four tools). Every run takes place in a bubblewrap sandbox that exposes the data space read-only, its own working folder writable, and no network except the model gateway. The data agent cannot see reference programs, graders or other queries.

**Runs.** A request refused by the model provider is retried in the same session. Only runs on the released version of each query enter the tables, and we check in every session log that the run was served by the intended model.

**External memory systems.** Mem0 (open-source, Chroma vector store) and Zep (open-source engine Graphiti, embedded Kuzu graph) receive the work log of the data agent after every query that builds experience: its notes while working, the commands it ran with the first lines of their output, and its final report. Before a target query they are searched with the query text, and the top ten items are added under a heading of their own. The extraction model is GPT-5.6-Luna (OpenRouter `openai/gpt-5.6-luna`, $0.2 / $0.02 / $1.2 per million tokens) for every model under test. Its cost, $1.69 over 1,157 extraction calls, is included in the cost per query of Mem0 and Zep.

**Statistical procedure.** Inferential comparisons in Table 22 are paired on the query: the two settings run on the same target query with the same seed. When a setting has three seeds, the accuracy of a query is first averaged over its seeds. Confidence intervals are hierarchical bootstrap intervals (10,000 replicates) that first resample tests and then the paired differences within each test. $p$-values are computed from the bootstrap distribution and adjusted with the Holm procedure within Table 22. Table 22 lists the tests behind the main claims with the smallest difference each design can detect.

**Aggregation.** For understanding, accuracy is first averaged over the target queries in each test and then over tests. LG is the difference of these per-test means between relevant and unrelated experience, Gap is that difference divided by the oracle difference, and Neg. is the fraction of tests where relevant experience lowers accuracy. New, Recur, and Retain average over queries; Joint

Table 14: Long sequences per case (GPT-5.6-Sol, Codex, Full). /: not applicable.

| | No memory | | | | Notes | | | | Notes + programs | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Case** | Acc | Tools | $/req | Programs built | Acc | Tools | $/req | Programs built | Acc | Tools | $/req | Programs built |
| AGR-JIT-01 | 1.00 | 10 | 0.22 | / | 1.00 | 7 | 0.23 | no | 1.00 | 7 | 0.23 | yes |
| AUT-JIT-01 | 0.89 | 21 | 0.75 | / | 1.00 | 11 | 0.36 | no | 1.00 | 8 | 0.28 | yes |
| BK-JIT-01 | 1.00 | 15 | 0.43 | / | 1.00 | 11 | 0.51 | no | 1.00 | 8 | 0.34 | yes |
| NYC-JIT-01 | 1.00 | 18 | 0.61 | / | 1.00 | 12 | 0.42 | no | 1.00 | 8 | 0.27 | yes |
| PW-JIT-01 | 1.00 | 15 | 0.41 | / | 1.00 | 8 | 0.30 | no | 1.00 | 8 | 0.26 | yes |
| PW-JIT-02 | 1.00 | 15 | 0.47 | / | 0.89 | 12 | 0.47 | yes | 0.89 | 10 | 0.35 | yes |

Table 15: Results per data space (GPT-5.6-Sol, Codex, Full).

| **Data space** | **Chains** | **Underst.** | **Changes** | **Cold** | **Unrel.** | **With exp.** | **LG** | **$/solve** |
|---|---|---|---|---|---|---|---|---|
| eu_electricity_market | 4 | 3 | 1 | 0.33 | 0.50 | 0.83 | +0.33 | 2.00 |
| ipeds | 3 | 3 | 2 | 0.50 | 0.50 | 0.83 | +0.33 | 0.92 |
| ny_labor_market | 2 | 5 | 0 | 0.63 | 0.53 | 0.73 | +0.20 | 1.23 |
| nyc_urban_mobility | 2 | 4 | 1 | 0.21 | 0.50 | 0.67 | +0.17 | 2.03 |
| ofac_sanctions | 2 | 5 | 0 | 0.60 | 0.50 | 0.70 | +0.20 | 0.82 |
| steam_public_user_behavior | 2 | 3 | 1 | 0.33 | 0.00 | 0.67 | +0.67 | 0.66 |
| terminal_search_ad_recommendation | 3 | 6 | 2 | 0.69 | 0.75 | 0.75 | 0.00 | 0.32 |
| uk_gcloud_saas | 3 | 4 | 1 | 0.75 | 0.88 | 0.88 | 0.00 | 2.62 |
| us_agriculture_commodities | 3 | 6 | 2 | 0.83 | 0.83 | 0.67 | −0.17 | 0.38 |
| us_automotive_quality | 3 | 3 | 1 | 1.00 | 1.00 | 0.67 | −0.33 | 2.26 |
| us_aviation_ops | 2 | 6 | 2 | 0.28 | 0.17 | 0.50 | +0.33 | 0.66 |
| us_mortgage | 2 | 5 | 2 | 0.65 | 0.45 | 0.50 | +0.05 | 1.82 |
| us_power_sector | 2 | 3 | 0 | 1.00 | 0.67 | 1.00 | +0.33 | 0.35 |
| us_respiratory_surveillance | 3 | 5 | 2 | 0.50 | 0.60 | 0.60 | 0.00 | 2.01 |
| us_retail_banking_supervision | 1 | 2 | 2 | 1.00 | 1.00 | 1.00 | 0.00 | 0.75 |

averages over tests. Reuse chains use three seeds. All understanding columns (Tables 3, 4, 18, and 21) use seed 1; Table 20 reports three-seed results separately on the queries measured in all three seeds. The B0–B3 rows of Table 4 average over three seeds; all other change-test rows use seed 1 unless stated otherwise. Each memory row in Table 4 uses the tests that its setting completed (Table 17), and LG and Gap use controls on that same set; Table 18 restricts every row to the 62 tests shared by all settings. Paired analyses average differences over matched target queries after averaging seeds, so their effects can differ from the test-weighted means. Values are rounded after aggregation.

## C Harness Details

All harnesses run with the same setting: the same solver prompt (Appendix K), the same read-only workspace and sandbox, the same budget of 40 minutes per query, and high reasoning effort where the harness exposes it. Every harness writes the same submission file, which the same grader scores.

- **Codex.** `codex exec` with reasoning effort *high*, rooted at the solve directory. Codex's own sandbox is disabled inside the bubblewrap sandbox, which keeps the data space read-only and blocks the network except the model gateway.
- **Hermes Agent.** The Hermes agent inside a bubblewrap mount namespace that exposes only the workspace (read-only) and its own run directory; its home directory is redirected so no persistent memory or skills leak in; browser and messaging tools are removed.
- **Pi.** The Pi CLI driven with the same prompt through an OpenAI-compatible gateway, with extensions, skills, context files and prompt templates disabled.
- **LangChain-DeepAgent.** The LangChain DeepAgents framework as shipped (planning, file and shell tools, subagents) with a local shell backend rooted at the solve directory.
- **Spider-Agent.** The open-source data agent released with Spider 2.0 (spider-agent-lite). Its own system prompt, text-parsed action space (bash, create file, edit file, database query, terminate), history window and repeated-action guard are used unchanged. Two things differ from the release: commands run in the same sandbox as the other harnesses instead of a Docker container, and the model is called through the same OpenAI-compatible gateway. Its parser accepts one text format per action and ends a run after four responses it cannot parse. This rule ends 54 %

Table 16: Interaction and cost per data space understanding query (Lite, Codex).

| | Without experience (C0) | | | With structured memory (C2) | | |
|---|---|---|---|---|---|---|
| **Model** | Tool calls | Tokens (k) | $/query | Tool calls | Tokens (k) | $/query |
| GPT-6-Astra | 16 | 391 | 1.42 | 7 | 146 | 0.63 |
| GPT-6-Sol | 10 | 218 | 0.22 | 7 | 127 | 0.13 |
| GPT-6-Luna | 10 | 241 | 0.026 | 7 | 137 | 0.016 |
| GPT-5.6-Sol | 18 | 483 | 0.59 | 8 | 176 | 0.28 |
| DeepSeek-V4-Pro | 32 | 879 | 2.11 | 11 | 176 | 0.54 |
| DeepSeek-V4.1-Flash | 27 | 583 | 0.024 | 8 | 125 | 0.0089 |
| Gemini-3.1-Pro | 34 | 761 | 0.78 | 15 | 247 | 0.41 |
| GLM-5.3 | 11 | 208 | 0.17 | 6 | 107 | 0.068 |
| GLM-5.3-Flash | 13 | 291 | 0.025 | 7 | 132 | 0.010 |
| Grok-4.6 | 22 | 683 | 0.65 | 11 | 237 | 0.30 |
| HY4-Preview | 27 | 469 | 0.094 | 9 | 150 | 0.033 |
| Kimi-K3 | 16 | 239 | 0.32 | 6 | 89 | 0.12 |
| Qwen-3.7-Max | 17 | 282 | 0.18 | 7 | 122 | 0.11 |

Table 17: Run counts behind Table 4 (GPT-5.6-Sol, Full, seed 1).

| **Setting** | **Tests** | **Queries** | **No answer** |
|---|---|---|---|
| No experience (C0) | 64 | 108 | 0 |
| Unrelated experience (C1) | 64 | 108 | 0 |
| Unrelated, full log | 63 | 105 | 16 |
| Session logs | 63 | 105 | 4 |
| Session logs, full log | 63 | 105 | 13 |
| Structured memory | 63 | 105 | 0 |
| Structured memory, K removed | 63 | 105 | 0 |
| Mem0 | 63 | 104 | 0 |
| Zep | 62 | 100 | 0 |
| Oracle (C3) | 64 | 108 | 0 |
| Old experience (B0) | 19 | 38 + 19 | 1 |
| Session logs (B1) | 19 | 38 + 19 | 0 |
| Structured memory (B2) | 19 | 38 + 19 | 0 |
| Oracle (B3) | 19 | 38 + 19 | 0 |
| Session logs, full log (B1) | 19 | 38 + 19 | 17 |
| Mem0 (B1) | 19 | 38 + 19 | 6 |
| Zep (B1) | 19 | 38 + 19 | 4 |

and 53 % of the runs of DeepSeek-V4.1-Flash and GLM-5.3-Flash, and 91 % of those of Kimi-K3. Kimi-K3 departs from the required format in 20 % of its responses (7–9 % for the other two models), and 7 % of its responses reach the 2,500-token output limit before the action. Harnesses with structured tool calls do not have this failure.

- **ReAct-based data agent.** A single chat-completions loop (Yao et al., 2023) with four tools (read-only SQL, read file, run Python, write file under the solve directory), and no planning. Most data agent baselines take this form (Lei et al., 2025; Huang et al., 2024; Hu et al., 2024; Majumder et al., 2025; Gu et al., 2024), and it is the simplest harness we test.

# D Evaluation Details

## D.1 Settings for data space understanding

A later query may go better for three reasons: the data agent has worked in the data space before, it recovered the relevant knowledge from its own experience, or the knowledge was given to it directly. We separate these reasons with four settings that change one factor at a time (Table 25). Every target query starts a fresh session in every setting. A difference between settings can therefore only come from what the data agent carries into the session.

Table 18: Table 4 on the common subset (GPT-5.6-Sol, Full, seed 1).

| **Understanding** | **Tests** | Use ↑ | LG ↑ | Gap ↑ | Neg. ↓ |
|---|---|---|---|---|---|
| No experience (C0) | 62 | 0.61 | / | / | / |
| Unrelated experience (C1) | 62 | 0.57 | 0.00 | 0.00 | / |
| Session logs | 62 | 0.63 | 0.06 | 0.22 | 0.15 |
| Session logs, full log | 62 | 0.67 | 0.10 | 0.36 | 0.11 |
| Structured memory | 62 | 0.72 | 0.14 | 0.52 | 0.08 |
| Structured memory, K removed | 62 | 0.65 | 0.08 | 0.30 | 0.11 |
| Mem0 | 62 | 0.70 | 0.13 | 0.46 | 0.08 |
| Zep | 62 | 0.68 | 0.11 | 0.40 | 0.08 |
| Oracle (C3) | 62 | 0.85 | 0.28 | 1.00 | 0.02 |
| **Data space changes** | **Tests** | New ↑ | Recur ↓ | Retain ↑ | |
| Old experience (B0) | 19 | 0.53 | 0.29 | 0.68 | |
| Session logs (B1) | 19 | 0.66 | 0.18 | 0.58 | |
| Structured memory (B2) | 19 | 0.74 | 0.03 | 0.58 | |
| Oracle (B3) | 19 | 1.00 | 0.00 | 0.74 | |
| Session logs, full log (B1) | 19 | 0.53 | 0.03 | 0.63 | |
| Mem0 (B1) | 19 | 0.53 | 0.21 | 0.63 | |
| Zep (B1) | 19 | 0.55 | 0.24 | 0.53 | |

Table 19: Whether the knowledge reaches the memory (GPT-5.6-Sol, Full, seed 1). K in memory: judged by GPT-6-Sol.

| | **Relevant / unrelated** | | | **K in memory** | | **Accuracy** | |
|---|---|---|---|---|---|---|---|
| **Setting** | History tool calls | Memory length | **Judged** | Stated | Applied | K in memory | K not in memory |
| Unrelated experience (C1) | / | / | 64 | 0.00 | 0.02 | / | / |
| Session logs | 0.92 | 0.91 | 63 | 0.05 | 0.16 | 0.69 | 0.60 |
| Session logs, full log | 0.92 | 1.00 | 63 | 0.48 | 0.25 | 0.74 | 0.53 |
| Structured memory | 1.00 | 3.39 | 63 | 0.62 | 0.16 | 0.76 | 0.50 |
| Structured memory, K removed | 1.00 | 3.50 | 63 | 0.00 | 0.02 | 0.00 | 0.66 |
| Mem0 | 1.17 | 1.55 | 104 | 0.35 | 0.20 | 0.84 | 0.49 |
| Zep | 1.11 | 2.15 | 100 | 0.21 | 0.17 | 0.87 | 0.52 |

**Matched unrelated experience.** The unrelated experience (C1) is matched to the relevant experience (C2). It has the same number of queries from the same data space. None of its queries shares evidence with the knowledge under test, and none is the source or target of any test. Its queries are chosen to need about the same interaction as the source query (median tool calls without experience within a factor of 0.8–1.25 when a candidate exists); in the runs, the median ratio is 0.92–1.17 (Table 19). The oracle (C3) adds the knowledge in one sentence to the C1 experience, without the answer (Appendix K). The matching is stored with every run.

### D.2 Settings for data space changes

For a data space change test, the data agent first solves the queries before the change and keeps its experience. Four settings then start from the change-point query. B0 skips the change point and answers the remaining queries with the old experience. B1 solves the change point and keeps session logs. B2 solves the change point and may update its structured memory. B3 receives the new knowledge in one sentence. Mem0 and Zep follow B1 with their own store. B0 never sees the change; B1 and B2 both solve the change-point query, but B1 keeps the result as session logs while B2 writes it into structured memory. On the reference model, session logs (B1) lower the outdated answers from 0.25 to 0.18, and structured memory (B2) lowers them to 0.05 (Table 4). The queries after the change and the query before it each start from the experience of the setting.

Table 20: Accuracy per seed (GPT-5.6-Sol, Codex, Full), averaged over the $n$ queries measured in all three seeds.

| **Cell** | $n$ per seed | Seed 1 | Seed 2 | Seed 3 | Mean | Range |
|---|---|---|---|---|---|---|
| Reuse chains, run 1 | 37 | 0.35 | 0.41 | 0.41 | 0.39 | 0.05 |
| Reuse chains, run 3 | 37 | 0.51 | 0.49 | 0.43 | 0.48 | 0.08 |
| Understanding, no experience (C0) | 76 | 0.47 | 0.47 | 0.47 | 0.47 | 0.00 |
| Understanding, unrelated experience (C1) | 108 | 0.52 | 0.60 | 0.61 | 0.58 | 0.09 |
| Understanding, structured memory (C2) | 105 | 0.70 | 0.72 | 0.70 | 0.71 | 0.03 |
| Understanding, oracle (C3) | 74 | 0.86 | 0.81 | 0.84 | 0.84 | 0.05 |
| Data space change tests, old experience (B0), after the change | 38 | 0.53 | 0.58 | 0.61 | 0.57 | 0.08 |
| Data space change tests, session logs (B1), after the change | 38 | 0.66 | 0.71 | 0.66 | 0.68 | 0.05 |
| Data space change tests, structured memory (B2), after the change | 38 | 0.74 | 0.66 | 0.68 | 0.69 | 0.08 |
| Data space change tests, oracle (B3), after the change | 38 | 1.00 | 0.95 | 0.95 | 0.96 | 0.05 |
| Data space change tests, old experience (B0), before the change | 19 | 0.68 | 0.58 | 0.63 | 0.63 | 0.11 |
| Data space change tests, structured memory (B2), before the change | 19 | 0.58 | 0.79 | 0.63 | 0.67 | 0.21 |

Table 21: Lite and Full for three models (Codex, structured memory). Reuse: three seeds; understanding and changes: seed 1.

| | | | **Understanding** | | | **Reuse** | | **Data space changes (B2)** | |
|---|---|---|---|---|---|---|---|---|---|
| **Model** | **Tests** | $n$ | Cold | Use | LG | Acc@1 | Acc@3 | New | Retain |
| GPT-5.6-Sol (reference) | Lite | 27 | 0.43 | 0.65 | 0.22 | 0.96 | 0.96 | 0.63 | 0.38 |
| GPT-5.6-Sol (reference) | Full | 63 | 0.60 | 0.70 | 0.13 | 0.39 | 0.48 | 0.74 | 0.58 |
| GPT-6-Sol | Lite | 27 | 0.44 | 0.68 | 0.22 | 1.00 | 1.00 | 0.63 | 0.63 |
| GPT-6-Sol | Full | 59 | 0.60 | 0.69 | 0.05 | 0.38 | 0.48 | 0.65 | 0.71 |
| DeepSeek-V4.1-Flash | Lite | 27 | 0.59 | 0.82 | 0.26 | 1.00 | 1.00 | 0.75 | 1.00 |
| DeepSeek-V4.1-Flash | Full | 59 | 0.65 | 0.78 | 0.15 | 0.41 | 0.54 | 0.71 | 0.82 |

### D.3 METRIC DEFINITIONS

A query is solved when every deterministic grading item passes, and accuracy over a set of queries is the share of solved queries. The cost $c(q)$ of a query is in USD at list price, with cached input priced at the cached rate.

- **Procedural reuse.** For a chain solved in one session, $\mathrm{Acc@}k$ is the accuracy of the $k$-th run, and $\mathrm{Tool@3}$ is the median ratio of the tool calls of the third run to those of the first run.
- **Data space understanding.** For a test with target query $q$, Cold is the accuracy under C0 and Use the accuracy under C2. The learning gain is $\mathrm{LG} = \mathrm{Acc}_{C2}(q) - \mathrm{Acc}_{C1}(q)$. Gap is the mean LG over tests divided by the mean oracle gain $\mathrm{Acc}_{C3}(q) - \mathrm{Acc}_{C1}(q)$ over the same tests, that is, the share of the oracle gain that the data agent recovers from its own experience. Taking the ratio of the means keeps a test with a very small oracle gain from dominating. Neg. is the share of tests with $\mathrm{LG} < 0$.
- **Data space changes.** New is the accuracy on the queries after the change. Recur is the share of these queries whose answer contains the value that the outdated knowledge produces. Retain is the accuracy on the query before the change. Joint is the share of tests whose queries after the change and query before it are all correct.
- **Cost.** \$/q is the mean cost per query, and \$/solve is the total cost divided by the number of solved queries. In Table 3 both are taken over the chains, the understanding targets under C2, and the queries of data space change tests under B2. The accuracy with experience is the mean of $\mathrm{Acc@3}$, Use, and New.
- **Long sequences.** Observed and held-out accuracy are over requests 1–6 and 7–10. Tool calls are medians and USD are means per request. "Builds programs" counts the sequences whose carried folder holds at least one program at the end.

Table 22: Paired tests behind the main claims (GPT-5.6-Sol, Full).

| Claim | Comparison | $n$ pairs | $\Delta$ | 95 % CI | $p$ (Holm) | Detectable $\Delta$ |
|---|---|---|---|---|---|---|
| Relevant experience raises accuracy | relevant vs. unrelated experience | 105 | $+0.12$ | $[0.05, 0.21]$ | 0.01 | 0.12 |
| Structured memory beats session logs | structured memory vs. session logs | 105 | $+0.10$ | $[-0.01, 0.21]$ | 0.13 | 0.16 |
| Reuse keeps accuracy | third vs. first query | 111 | $+0.09$ | $[-0.01, 0.20]$ | 0.08 | 0.15 |
| Updating memory helps after the change | updated vs. old experience | 38 | $+0.12$ | $[0.00, 0.25]$ | 0.13 | 0.18 |

Table 23: Provenance of each result table.

| Table | Tier | Tests | Seeds | Aggregation |
|---|---|---|---|---|
| Table 3 (main results) | Lite | 9 reuse, 27 underst., 7–8 changes | 3 (reuse), 1 (others) | Per test, then avg. over tests |
| Table 4 (memory) | Full | 62–64 underst., 19 changes | 1 (underst.), 3 (B0–B3 changes), 1 (other changes) | Underst.: per test; changes: per query |
| Table 18 | Full | 62 underst. (common subset) | 1 | Per test, then avg. over tests |
| Table 21 | Both | Lite and Full | 1 (underst., changes), 3 (reuse) | Per test, then avg. over tests |
| Table 22 | Full | As listed per row | See Appendix B | Paired bootstrap (10k replicates) |

### D.4 GRADING

Each query has an answer contract and a grading file. Deterministic items compare the submitted field with the reference: numeric with an absolute or relative tolerance (3,254 items over the 742 queries), field equality (648), exact list comparison (147), and a small number of set, non-empty, file, findings and executed-check items (55). Lists of records are compared by value regardless of key order. A query is solved only when every deterministic item passes, including the critical ones. Every reported accuracy uses these deterministic items only. Each query also comes with six to ten binary rubric items (three types, one to three critical) that describe the expected process and result. They are released for process-level analysis.

### D.5 MEMORY MECHANISMS

The queries that build experience are solved in order, each in its own session. Between queries the memory mechanism keeps one of the following. (1) Nothing (no experience). (2) Session logs placed before the target: either the final message of each session or its full log with commands, outputs and messages (at most 120,000 characters in total; the middle of long sessions is elided). (3) Structured memory: the data agent's own `notes.md`, which it rewrites after every query, plus an `annex/` folder of rules and programs it chose to keep (Zhao et al., 2024; Wang et al., 2024; 2025), given read-only to the target. (4) The items that Mem0 or Zep retrieve (Appendix B). The experience after the last of these queries is frozen with a content hash. Every target of a test starts from that frozen state in a fresh session, and target sessions are never shown to another target. The cost of a target query includes reading and retrieving experience. The cost of building that experience (the source sessions) is reported separately in Appendix A.9. The cost of Mem0 and Zep extraction calls is included as described in Appendix B.

### D.6 SAME-SESSION CHAINS

A short reuse chain runs as one session of the harness. The first query starts the session, and each later query resumes the same session with a short header added before the solver prompt (Appendix K). Each query has its own submission folder, answer file, grade and cost. The first run receives no experience of any kind. Task attachments are copied into the session folder before each query.

### D.7 LONG SEQUENCES

Long sequences run under five settings with the same requests in the same order: (1) no memory, where the state is cleared before every request; (2) notes kept between requests (Shinn et al., 2023); (3) notes plus programs, where the data agent is asked to build and reuse executable programs under `annex/`; (4) notes with the programs removed before the held-out requests, which starts from the notes-plus-programs state after the sixth request; (5) a prompt to build programs with no memory. After the runs we check every request folder. A program counts as built when a new executable, parameterized script appears under `annex/`, and as used only when a held-out request executes it.

Table 24: Models and list prices (USD per million tokens: input / cached / output).

| Model | OpenRouter identifier | Vendor | Price |
|---|---|---|---|
| GPT-6-Astra (OpenAI, 2026b) | `openai/gpt-6-astra` | OpenAI | 10 / 1 / 50 |
| GPT-6-Sol (OpenAI, 2026b) | `openai/gpt-6-sol` | OpenAI | 2 / 0.2 / 10 |
| GPT-6-Luna (OpenAI, 2026b) | `openai/gpt-6-luna` | OpenAI | 0.1 / 0.01 / 0.5 |
| GPT-5.6-Sol (OpenAI, 2026a) | `openai/gpt-5.6-sol` | OpenAI | 4 / 0.4 / 20 |
| DeepSeek-V4-Pro (DeepSeek-AI, 2026b) | `deepseek/deepseek-v4-pro` | DeepSeek | 1.74 / 0.145 / 3.48 |
| DeepSeek-V4.1-Flash (DeepSeek-AI, 2026a) | `deepseek/deepseek-v4.1-flash` | DeepSeek | 0.15 / 0.003 / 0.6 |
| Gemini-3.1-Pro (Google DeepMind, 2026) | `google/gemini-3.1-pro-preview` | Google | 2 / 0.2 / 12 |
| GLM-5.3 (GLM-5-Team, 2026) | `z-ai/glm-5.3` | Z.ai | 1.4 / 0.26 / 4.4 |
| GLM-5.3-Flash (Z.ai, 2026) | `z-ai/glm-5.3-flash` | Z.ai | 0.15 / 0.03 / 0.5 |
| Grok-4.6 (xAI, 2026) | `x-ai/grok-4.6` | xAI | 2 / 0.5 / 6 |
| HY4-Preview (Tencent Hy Team, 2026) | `tencent/hy4-preview` | Tencent | 0.834 / 0.042 / 2.501 |
| Kimi-K3 (Kimi Team, 2026) | `moonshotai/kimi-k3` | Moonshot | 3 / 0.3 / 15 |
| Qwen-3.7-Max (Qwen Team, 2026) | `qwen/qwen3.7-max` | Alibaba | 1.475 / 0.295 / 4.425 |

Table 25: The four settings for data space understanding. Each row changes one factor.

| Setting | What the data agent carries into the session | Contrast and the effect it identifies |
|---|---|---|
| C0 | No experience | Reference point |
| C1 | Unrelated experience from the same data space | C1−C0: effect of other work in the data space |
| C2 | Experience from the same number of queries that contains the knowledge | C2−C1: learning gain from relevant experience |
| C3 | The C1 experience plus the knowledge as a verified statement | C3−C1: oracle gain; $(C2-C1)/(C3-C1)$: gap closure |

### D.8 Cost, time limits and seeds

Cost is recorded per query as tokens, tool calls, SQL queries, files opened, code runs and wall time. Tokens are read from the session records of every run and split into uncached input, cached input and output. Dollar figures price each class at the undiscounted OpenRouter list price of the model, retrieved on 2026-09-24 (Table 24). We also report the cost the same tokens would have without cache hits (Table 9). Every run has the same budget of 40 minutes. A run that exceeds the limit counts as a failure, and its cost is kept. Reuse chains use three seeds. On the reference model, the understanding settings C0, C1, C2 (structured memory) and C3 and the settings B0–B3 of the data space change tests also use three seeds; every other cell uses one seed.

### D.9 Version lock

Every query carries a version record with separate hashes of its question and contract, grading, reference program, and attachments. Every run stores the version it was produced on. A run whose version differs from the released query is discarded and repeated. The review console stores the same version with every human record.

## E Examples

**Reuse chain (us_retail_banking_supervision).** Knowledge: the six macro series in `fact_bank_macro_series` have different date histories (CORCCACBS from 1985-01, DRCCLACBS from 1991-01, DRCLACBS from 1987-01, REVOLSL from 1968-01, TDSP from 2005-01, TOTALSL from 1943-01); only 85 of 1,000 dates carry all six, retaining 510 of 2,248 rows. Run 1: "The briefing needs macro context without hiding the different histories of the six series. Summarize each series' available period and observation count, report how many dates support all six series, and recommend the denominator for a synchronized comparison in the coverage note." Contract: `series_periods`, `common_dates`, `recommended_denominator`, `series_count`, `long_format_observations`, `common_panel_observations`, `common_panel_retention_pct`; reference answer: 85 common dates, 510 of 2,248 rows (22.69 %). Run 2: the same routine on 2010–2020 ("prepare a stress-monitoring ap-

pendix ...covering 2010 through 2020"), contract `series_periods`, `common_dates`, `total_observations`, `common_panel_coverage`. Run 3 repeats it for 2005-01 to 2026-04. Rubric items include, for run 2, "the retention fraction reflects only observations on dates represented by all six series" (critical).

**Understanding test (uk_gcloud_saas).** Knowledge (semantic): "The G-Cloud 10 agreement defines a 24-month initial call-off term with two optional 12-month extensions, giving a 48-month maximum; `fact_procurement_award.duration_months` is the numeric duration field to compare with that cap." The rule is stated in the framework agreement PDF of the workspace, not in the schema. Source (unscored): "The contracts team is testing duration exceptions in an archived G-Cloud 10 award period covered by the old agreement. Use that governing agreement to classify the full term, retain notice-level values, and show the effect of extensions on the exception queue..." Target (scored): "Service Transition Finance is costing a term-remediation exercise for the H M Revenue & Customs buyer entry in the 2024 Q4 award extraction. Apply the G-Cloud 10 call-off term benchmark to this portfolio. Estimate the total months that need trimming and the largest individual reduction, allowing £900 per month trimmed..." Without the knowledge, the trimmed months and costs are wrong, and the reference program stores these values as the wrong answer.

**Long sequence NYC-JIT-01 (nyc_urban_mobility).** Deliverable: a monthly yellow-taxi summary with mileage totals and the three pickup zones with the most trips, each with its share of all yellow trips, under one fixed answer contract. Observed run 1: "Before closing the January 2023 ledger, yellow-taxi operations needs a mileage and demand check. Summarize distance efficiency and show which three pickup zones led the month, including their portions of all yellow rides." Held-out run 7: the same deliverable for March 2025, "A yellow-cab service manager is reconciling the March 2025 operating records before signing the monthly operating review..." A fresh start needs 18 tool calls per request on this case; notes plus programs need 8.

**Data space change test TSA-REV-01-P (terminal_search_ad_recommendation).** The IAB Content Taxonomy is the code list that ad platforms use to label content. In release 2.2, category 150 is Events and Attractions with 34 leaf categories. In release 3.0, it becomes Attractions with 11 leaf categories. Both releases are in the workspace. The two queries before the change size the test workload of an Events and Attractions picker under release 2.2. The query at the change point asks to reconcile the current Attractions picker with the archived one, and with the old release it reports 34 choices instead of 11. The two queries after the change size the test workload of the current picker (e.g., four locales and three device types at seven minutes per case) and are correct only with 11 choices. The retention query rebuilds the test workload of an archived campaign and must still use the 34 choices of release 2.2.

## F Benchmark Statistics and Data Spaces

A workload is a sequence of queries over one data space. The 742 queries are organized into 84 workloads across the 15 data spaces. Table 26 gives the full statistics of both tiers. DataIntern uses 15 data spaces, each a snapshot of real public data built by an automated collection pipeline (68 GB of DuckDB databases, raw files and documents in total). A data agent reaches a data space only through a read-only workspace: a DuckDB database, the curated and raw dataset files, the reference documents, the workspace's own ETL code and schema notes, and operational logs. Table 27 lists the data spaces and their queries by scenario.

**DataIntern-Full and DataIntern-Lite.** DataIntern-Full is built from enterprise feedback to simulate a realistic workload, covering all query types that a data agent would encounter. DataIntern-Lite extracts the subset of tests that specifically require learning, where the query needs knowledge from the data space and where experience can change the outcome. Experts select Lite queries from the full set based on three criteria: the query passes the release gate and scores at least 4 on naturalness, the test leaves clear room to learn (the oracle improves accuracy over no experience), and no data space dominates a scenario (at most two tests per data space).

Table 26: Full statistics of DataIntern-Full and DataIntern-Lite.

| Statistic | Full | Lite |
|---|---|---|
| *Environment* | | |
| Data spaces / domains | 15 / 6 | |
| Tables and views | 450 | |
| Columns | 6227 | |
| Documents | 13559 | |
| Data volume | 68 GB | |
| *Composition* | | |
| Queries | 742 | 193 |
| Reuse chains / long sequences | 37 / 6 | 9 / 5 |
| Understanding tests (queries) | 64 (170) | 27 (72) |
| Data space change tests (queries) | 19 (110) | 8 (48) |
| Background and controls | 287 | 0 |
| Knowledge units under test | 83 | 35 |
| *Request and answer contract* | | |
| Words per request | 43.8 | 45.6 |
| Answer fields per query | 5.1 | 5.3 |
| Deterministic checks per query | 5.5 | 6.3 |
| Critical checks per query | 2.7 | 2.5 |
| Rubric items per query | 6.8 | 7.1 |
| Queries with attachments | 31 | 1 |
| *Difficulty* | | |
| Cold-start pass rate (reference model) | 0.63 | 0.69 |
| Tool calls per cold solve (median) | 16 | 16 |

Table 27: The 15 data spaces and their queries by scenario.

| Data space | Domain and period | Tables | Files | Docs | Queries | Reuse | Underst. | Changes | Backgr. |
|---|---|---|---|---|---|---|---|---|---|
| eu_electricity_market | EU hourly electricity market, 10 zones, 2023-04 to 2026-03 | 68 | 1150 | 156 | 42 | 12 | 8 | 6 | 16 |
| ipeds | US higher education (IPEDS), 2019–2024 | 21 | 231 | 44 | 42 | 9 | 7 | 12 | 14 |
| nyc_urban_mobility | NYC taxi and for-hire trips, subway, weather, 2023–2025 | 19 | 361 | 169 | 63 | 15 | 11 | 6 | 31 |
| ny_labor_market | New York State labor statistics, 2018–2026 | 41 | 205 | 467 | 43 | 9 | 15 | 0 | 19 |
| ofac_sanctions | US OFAC sanctions actions and lists, 2015–2026 | 33 | 301 | 730 | 39 | 7 | 13 | 0 | 19 |
| steam_public_user_behavior | Steam reviews, catalog, hardware surveys | 32 | 353 | 86 | 38 | 6 | 8 | 6 | 18 |
| terminal_search_ad_recommendation | Mobile search advertising and content recommendation | 13 | 75 | 151 | 46 | 9 | 15 | 10 | 12 |
| uk_gcloud_saas | UK G-Cloud public-sector SaaS procurement, 2018–2026 | 28 | 764 | 770 | 42 | 9 | 10 | 6 | 17 |
| us_agriculture_commodities | USDA crop and livestock statistics, 2022–2026 | 33 | 1083 | 110 | 65 | 16 | 16 | 12 | 21 |
| us_automotive_quality | Vehicle recalls, ratings, plants and suppliers | 28 | 150 | 234 | 50 | 18 | 7 | 6 | 19 |
| us_aviation_ops | US airline on-time performance and traffic, 2023–2026 | 14 | 180 | 573 | 59 | 6 | 21 | 12 | 20 |
| us_mortgage | US mortgage origination, portfolios, regulation | 31 | 78 | 5,149 | 67 | 12 | 16 | 12 | 27 |
| us_power_sector | US plant operations, emissions, finance, 2018–2025 | 41 | 325 | 95 | 47 | 25 | 7 | 0 | 15 |
| us_respiratory_surveillance | CDC flu, COVID and RSV surveillance, 2023–2026 | 27 | 367 | 190 | 50 | 9 | 12 | 11 | 18 |
| us_retail_banking_supervision | CFPB complaints, FDIC financials, regulations, 2018–2026 | 21 | 56 | 4,635 | 49 | 13 | 4 | 11 | 21 |

# G Benchmark Design Principles

The design of DataIntern is guided by three principles: the knowledge is native to the data, the relations between queries are verified by execution, and every query remains independent.

**Native Knowledge.** A benchmark of learning is meaningful only if the knowledge to be learned is real. We achieve this through two choices. (1) **Real sources:** every data space is built from public data and its official documents, and the data quality problems of the sources are kept as they are. (2) **No simulated change:** data space change tests use condition changes that the sources already contain, such as a fee that starts in January 2025 or a code vocabulary that a later release replaces. Enterprise feedback decides which kinds of knowledge we look for, but it never supplies the knowledge itself.

**Execution-Verified Relations.** Two related queries do not by themselves form a learning opportunity. We therefore verify each relation by execution. (1) **Executable references:** every answer field is computed by a reference program at run time, without literals. (2) **Traced evidence:** the evidence of a query is what its reference program reads at run time (Appendix I.8). It decides whether a source query exposes the knowledge and which earlier queries count as unrelated. (3) **Wrong answers:** the reference program also computes the answer obtained without the knowledge (Appendix I.7). A wrong answer that differs from the correct one shows that the answer of the target query depends on the knowledge. We also check with real data agents that each test leaves room to improve (Appendix I).

**Independent Queries.** Experience should help a data agent, but it should never be required. (1) Every query is solvable from the data space alone. (2) No query needs the answer of another query, and the answer of a target query cannot be copied from its source query. (3) A target query is scored only when a data agent without experience leaves room for improvement in accuracy or in cost. These conditions allow us to compare the same data agent with and without experience on the same query.

# H ENTERPRISE FEEDBACK

The feedback comes from three enterprise sources: execution logs, trajectories of data agents, and customer feedback. Construction agents cluster the items, and the experts summarize the clusters into the four kinds of knowledge and the workload objectives used in the design of each data space. The released feedback is anonymized: company names, customer identifiers and internal table names are removed. Table 28 gives three items of the feedback and the scenario each one informs.

Table 28: Examples of enterprise feedback (banking relationship managers, anonymized) and the scenario each one informs.

| **Feedback** | **Scenario** |
|---|---|
| "Mondays are the worst. I'm in at 8:30, coffee, and have until 9:00 to scan everything that's moved since the night before." The same review is repeated every week. | Procedural reuse |
| "Of the CNY 430M in tenders, 180M of that was a joint award where we held a 20 % share." The source data has a joint-versus-sole column, and the assistant summed over it. | Data space understanding |
| "Risk review took out Customer Z's CNY 8M two weeks ago." A week later the assistant still counted the removed amount, because what it remembers updates only within a conversation. | Data space changes |

# I CONSTRUCTION DETAILS

## I.1 WORKFLOW-FIRST SYNTHESIS

Every query in DataIntern is synthesized by an authoring agent against the real data and then verified by execution. For seven of the 15 data spaces the synthesis is workflow-first. A first session writes a *workflow plan* for the data space: a named team, a period of several months, 30–40 requests in time order, and at least six pieces of knowledge, each with the query that contains it and the wrong answer a careful analyst would get without it. The plan marks the role of every request: a source that contains a piece of knowledge, a repeat of the same routine with new parameters, a different routine whose answer is wrong without the knowledge, an observed or held-out request of a long sequence, or a background request of the same team. We check the plan before synthesis: request times never decrease, each source contains exactly one piece of knowledge, except two sources that each carry two pieces of knowledge, so 64 tests use 62 unique source queries, held-out requests appear nowhere else, no request needs the answer of another, and every piece of knowledge is specific to the data space (a generic method such as "check the join key" is rejected).

A second session then writes each request of the plan: the question in the requester's voice (no field lists and no rule stated), an answer contract with four to six named fields (at least one critical, enumerations with allowed values, and no field that only repeats a parameter), a reference program that computes every field from the data at run time without literals, the correct and wrong answer of each field, deterministic grading items, and a binary rubric of six to ten items. The other eight data spaces were built earlier, test by test, with the same files per query. Their workloads were then assembled from the tests and the background queries of the data space.

### I.2 TESTS

After synthesis, every test is assigned to exactly one scenario by how it is run (§2). A short reuse chain is one routine whose first query and its repetitions are solved in one session, scored on its first three runs. The release has 37 chains. Chains whose repetitions three model judges rated as not the same analysis were moved to the background. A long reuse sequence holds one recurring deliverable with six observed and three or four held-out requests (Appendix I.4). An understanding test is one piece of knowledge with its source query and one or more targets that need it in a different analysis. A data space change test is built around one condition change (Appendix I.3). A query is scored in exactly one scenario, and source queries are not scored. Queries that belong to no test are background queries (287, including the queries used as unrelated experience). The knowledge under test covers four types: structural, semantic, source-related (which source or version to trust), and procedural.

**Test certification.** Every understanding and data space change test is run without experience and with the oracle. A test is kept when the oracle improves accuracy or reduces tool calls. Tests where the target is already solved without experience, or where the oracle does not help, are removed.

### I.3 DATA SPACE CHANGE TESTS

A data space change test checks whether a data agent uses the right knowledge before and after one change of conditions in the data space, such as a new policy, a new data release, or a document that replaces an earlier one. These tests are built from condition changes that the data spaces already contain. Each test records the change, the old knowledge, the new knowledge, and the evidence files. It has four groups of queries: two queries before the change where the old knowledge holds, one change-point query on which the old knowledge produces a visible inconsistency, two queries after the change that need the new knowledge, and one retention query before the change. The change need not be a calendar date. A partition or source that follows a different convention, or a document that supersedes another, also qualifies, as long as the old knowledge was correct where it was formed. The 19 tests cover 12 data spaces and use 110 queries (most tests have 6 queries; four have 5 because they share a before-the-change query with another test). Examples include the Regulation C asset-size exemption ($47M for 2020, $58M for 2025); the G-Cloud call-off term (24 months with two extensions under G-Cloud 10, 36 months with one extension under G-Cloud 14 from 2024-10-29); the annual RFS percentage schedule of 2025; a French imbalance settlement period that changes to 15 minutes on 2025-01-01; the 2024 in-city driver-pay rate for New York for-hire vehicles; the 2024–25 influenza vaccine composition; the IAB Content Taxonomy 3.0 release; a WASDE report that revises an earlier estimate; and a vPIC release that renames a body class. Every test passes four execution checks. (1) The old knowledge gives the correct answer on the queries before the change and on the retention query. (2) It gives a wrong answer on the change-point query and the queries after the change. (3) Applying the new knowledge outside its scope gives a wrong answer on the retention query, while knowledge that records its scope is correct on both sides. (4) The evidence is in the workspace and is read by the reference program of the change-point query. We also check every test against the source documents, and against the official source where the rule is public. Each query must fall where the knowledge it needs applies.

### I.4 LONG REUSE SEQUENCES

The 6 long reuse sequences each hold one recurring deliverable with a fixed answer contract: six observed requests and three held-out requests (four for two cases), all with different parameters, requesters and wording, on 5 data spaces. Held-out answers were hidden while the requests were written and are pairwise distinct. The cases are AGR-JIT-01, AUT-JIT-01, BK-JIT-01, NYC-JIT-01, PW-JIT-01 and PW-JIT-02. Lite keeps all of them except PW-JIT-02.

### I.5 RELEASE GATE

Every released query passes all hard checks of an automated gate (Table 29). The gate runs after every repair round, and a failing query goes back to a rewrite session that may only edit what the failing check concerns. The loop repeats until every query passes. Two checks (P1, R2) are advisory and do not block. Human verification (Appendix J) covers what the automatic checks cannot.

Table 29: Release gate checks; a query passes only when every hard check passes.

| Check | Rule |
|---|---|
| Q1 natural | reviewer score $\geq 4$ on the naturalness rubric (Appendix I.9); the newest review of the current wording is used |
| Q2 no template | the question's sentence frame (entities, periods, numbers and field tokens masked) is shared by fewer than 3 queries; fewer than 4 snake_case tokens in the text |
| Q3 English | no CJK characters |
| Q4 no leak | no reference answer value in the question; no distinctive identifier of the tested knowledge in a target's question |
| Q5 no method leak | the reviewer did not tag `method_leak` (the request must not state the deciding rule, denominator or filter) |
| Q6 no contract leak | the answer contract does not restate the rule |
| R1 rubric present | 6–10 binary rubric items of three types (result, process, foundation) with 1–3 critical items |
| R2 rubric review (advisory) | no rubric item whose masked text is shared by $\geq 3$ queries; model review of judgeability |
| G1 grading integrity | $\geq 4$ deterministic items, $\geq 1$ critical, all items gating; the reference program is deterministic |
| G2 no free points | no answer field that is a single-value enumeration or echoes a request parameter |
| G3 reference runs | the reference program executes twice identically and matches the stored correct and wrong answers |
| G4 no placeholders | no contract field named like a stub (`metric_a`, `value_1`) |
| G5 contract well-formed | every field maps to a type, description and comparison rule |
| G6 no planted reads | no executed query whose result is unused and no comment listing evidence atoms in the reference program |
| D1 headroom | scored targets only: a cold-start seed fails or the mean cold score is below 0.7, or the cold-start median interaction is $\geq 25$ tool calls |
| P1 pair rule (advisory) | the three-judge overlap rating agrees with the scenario |
| P2, P3 | a target belongs to one test only and is not also a source |
| R3 contract | every request of a routine case returns the case's shared contract |

### I.6 Reference programs, independent solving and adjudication

Every reference program in the release runs twice and must match its stored answers (check G3, 742 of 742 pass). Before the release was frozen, an independent solver session solved every one of the 742 queries without seeing the reference program, and every disagreement was adjudicated independently. 601 of the 742 independent answers agree with the reference; the 141 disagreements were resolved as 29 wrong reference programs (fixed and solved again), 61 tolerance changes, 40 rewritten questions, and 11 correct references.

### I.7 Wrong answers

A wrong answer is the value that the reference program obtains when it ignores the knowledge of the query, for example when it adds the state surcharge to the CBD fee. It is stored next to the correct answer of the same field. We compare the two values field by field over the frozen release. A numeric field counts as different when the gap exceeds three times its grading tolerance. Of the 742 queries, 701 store a wrong answer and 700 of them differ from the correct answer. The remaining 41 queries store none. Among the 108 target queries of the understanding tests, 106 have a wrong answer that differs from the correct one and 2 have none. The tests of these two targets are kept because the oracle raises their accuracy in the certification runs (Appendix I).

### I.8 Evidence atoms

For every query we record the evidence its reference program actually uses: the DuckDB tables and columns in the SQL that the program executes (captured at run time, including shared modules and computed table names) and the files it opens (captured with `strace`). Only executed SQL counts, not table names in code or comments. These atoms decide which queries share evidence with a piece of knowledge. They are used to match the unrelated experience (Appendix D.1) and are shown to the experts during review.

### I.9 Model reviewers

Two model reviewers are used during construction. The naturalness reviewer (GPT-5.6-Sol) rates every question on a fixed rubric (Appendix K), and check Q1 requires a score of 4 or more. It sees only the question text and the names of the answer fields, never the data, the reference answer or any solver run, so it rates the writing of a question and not its difficulty. The procedure-overlap judge

rates how much of the source procedure a target repeats. It runs with three models (GPT-6-Astra, GPT-5.4, GPT-5.6-Terra), and the median rating is stored with every source–target pair.

### I.10 Models and cost of construction

Authoring, repair and scouting sessions run GPT-6-Astra in Codex with high reasoning effort (1,137 sessions between 2026-09-08 and 2026-09-17). Table 30 reports the cost of these sessions from their token records. The total corresponds to 6.7M tokens and $10.7 per released query. Most input tokens are cached, because a session reads the same data space many times. Repair after the release gate costs 2.6 times as much as the first synthesis: writing a query is cheap, and making it pass execution-based checks is not.

Table 30: Construction cost (1,137 Codex sessions; tokens in millions).

| Stage | Sessions | Uncached input | Cached input | Output | USD |
|---|---|---|---|---|---|
| Query, test and workload synthesis | 371 | 52.3 | 1,156.1 | 5.6 | 1,959 |
| Repair rounds after the release gate | 623 | 151.2 | 3,016.5 | 11.1 | 5,082 |
| Adjudication of reference programs | 20 | 6.8 | 265.6 | 0.9 | 379 |
| Data space change tests | 72 | 4.5 | 47.5 | 0.5 | 117 |
| Other (reviews, unreleased target queries) | 51 | 9.3 | 258.8 | 0.9 | 396 |
| Total | 1,137 | 224.2 | 4,744.4 | 18.9 | 7,932 |

## J Human Verification

Eight experts in agents and databases review the queries through a web console. For each query, a reviewer sees the request, the answer contract, the reference answer, the evidence the reference program reads, and the rubric with its reference answers. The reviewer approves the query, asks for a modification with a reason, or rejects it with a reason. Every record stores the version of the query it was made on, and a later edit of the query invalidates older reviews.

Every released query was reviewed by at least one expert. The experts made 2,338 decisions in the console, with 1,952 approvals, 366 modifications, and 20 rejections. On the 579 queries with two or more decisions, reviewers agree on the decision (approve, modify, reject) in 77 % of the pairs (Gwet's $AC_1$ 0.73).

## K Prompts

This section gives the prompts used in the evaluation and in construction. Text in angle brackets is filled in per query.

**Solver prompt.** Every harness receives the same solver prompt.

**Solver prompt, identical across harnesses**

```
You are a data analyst working in a data workspace you operate.

Workspace root (READ-ONLY -- treat as the entire universe of available data):
  <workspace_root>
It contains output/datasets/ (raw + curated files), output/documents/ (reference
documents), output/duckdb/<data_space>.duckdb, and operational assets: dashboard/ and
artifacts/ (published reporting), pipelines/ (the workspace's views and ETL code),
docs_schema/ (orientation docs), ops_log.jsonl, modeling.yaml, sources.yaml.
Query contract: connect with duckdb read_only=True and os.chdir to the workspace root
first (views use relative paths). Do not modify the workspace or use files outside it.

[## Notes from prior work -- the ONLY section that differs between conditions; see the
 history boxes below. Absent under the cold-start condition.]

## Task
<persona_context>
```

```
<question text of the task>

[Task-specific input files (copies, in your working directory):
  inbox/<file> ...]

## Deliverable
Write EXACTLY one JSON file at <solve_dir>/submission/answer.json of the form
{"task_id": "<task_id>", "answer": {"<field_1>": <number>, "<field_2>": "<string>", ...}}
The answer object must have EXACTLY these top-level keys, each mapped directly to its
value (plain scalars/lists -- no wrapper objects, no type/desc/value envelopes):
<answer_spec JSON: field -> {type, desc, unit, rounding, allowed_values}>
[Additionally produce these files (paths relative to your working directory): ...]

[## duties -- setting-specific, see the memory boxes]

Verify your numbers by re-querying before you write the file. Precision and
correct handling of the data matter more than speed.
```

**Experience settings.** Session logs, structured memory and the oracle add the following text before the target query. Same-session chains add a short header before each later query.

**Session logs: experience rendered before the target**

```
## Your final reports from earlier tasks in this workspace
### Final report of <earlier query 1>
<the final report the agent wrote for that earlier task, verbatim>

### Final report of <earlier query 2>
<...>

[If the concatenation exceeds 120,000 characters, the middle is replaced by
 "[...older transcript text elided...]".]
```

**Structured memory: the duty given with every query that builds experience, and what is rendered before the target**

```
# Duty appended to every query that builds experience:
After you finish the task, rewrite <run_dir>/notes.md so it records what you learned about
this workspace that would help next time (under 80 lines). Files you want to keep for later
use may be placed under ./annex.

# Rendered before the target query:
## Notes from your prior work in this workspace
You have worked in this workspace before. These are your own running notes:
<notes.md as frozen after the last query that builds experience>   [or: (no notes yet --
     this is early in your tenure)]

You also previously created reusable artifacts under ./annex in your working directory (
     listing):
./<file_1>
./<file_2>
You may read and query them.
```

**Oracle setting: the knowledge added after the unrelated experience**

```
## A verified fact about this workspace
<the knowledge statement of the test, e.g.: The yellow congestion_surcharge field is the
     New York State
 congestion surcharge and should not be combined with the separate CBD congestion fee when
 measuring that surcharge.>
```

**Same-session chain: header prepended to the second and third queries**

```
## A new request has arrived
The previous request is finished. Treat the following as a separate request from the same
     team;
write its answer to the path given below.
```

```
<the full solver prompt of the next task follows>
```

**Long sequences.** The duties of each setting and the state given before each request.

**Long sequences: duties per setting and the carried state rendered before each request**

```
# Rendered before every request (all settings except no memory and the program prompt alone
    ):
## Notes from your prior work in this workspace
<memory/notes.md>          [or: (no notes yet)]
Reusable artifacts under ./annex:
- <file_1>
- <file_2>

# Duty, notes setting:
After you finish the task, rewrite notes.md so it records what you learned about this
workspace that would help next time (under 80 lines). Files you want to keep for later
use may be placed under ./annex.

# Duty, notes + programs setting (also given to the programs-removed and program-prompt
    settings):
This workload repeats with varying parameters. You are encouraged to BUILD and REUSE
executable machinery: put parameterized scripts/queries under ./annex and invoke them on
later tasks instead of re-deriving the procedure. After solving, rewrite notes.md (under 80
lines) documenting how to run your annex tools.
```

**Construction reviewers.** The naturalness reviewer and the procedure-overlap judge (Appendix I.9).

**Question naturalness reviewer (GPT-5.6-Sol)**

```
You are reviewing questions of a data-analysis benchmark. Each task gives an analyst agent
    a
workspace of real public data files plus this question; the agent must produce a JSON
    answer whose
fields are fixed by a separate answer contract (you will see the field names). Judge ONLY
    the question
text as a piece of writing a real analyst, manager or auditor would send to a colleague.

Score 1-5:
5 = reads like a genuine request: a purpose or situation, a clear ask, natural prose; the
    deliverable is
    described in words, not as a glued list of output field names.
4 = natural and clear with one minor stiffness.
3 = understandable but formulaic: a template opening ("For X, report a, b, c, d, e...") or
    the deliverable
    is essentially the field list read aloud.
2 = mostly a concatenation of field names / spec fragments; no real purpose; awkward or
    robotic.
1 = unreadable, contradictory, not English, or unanswerable from the wording.

Defect tags (list all that apply):
- field_list_concat: the sentence is a comma-glued list of output fields (>=5 items)
    instead of prose
- template_opening: starts with "For <slice>, report/audit ..." with no situation or
    purpose
- no_purpose: no reason/situation for the request
- hidden_enum: asks for a category/decision but the allowed labels are not stated and not
    obvious
- vague_referent: refers to "the supplied/documented/requested" thing without naming it
- spec_tone: reads like a specification or grader instruction, not a human request
- non_english: any non-English text
- too_long_list: more than 8 separate items requested in one sentence
- unnatural_phrase: wording no analyst would use (quote it in the note)
- method_leak: the request spells out the deciding rule, denominator, filter or field
    definition that a competent analyst would be expected to find or verify from the data
    and documents themselves (e.g. "use fact rows as the denominator so code coverage does
     not overstate"); a real manager states the goal, not the recipe
```

```
Return STRICT JSON: {"score": int, "defects": [tags], "note": "<=40 words", "rewrite_hint":
      "<=60 words describing how a human would phrase this request (do not change what is
     asked)"}
```

**Procedure-overlap judge; run with GPT-6-Astra, GPT-5.4 and GPT-5.6-Terra, median used**

```
You audit a benchmark of data-analysis tasks. You are given a SOURCE task the agent already
      solved, the reusable
experience K it could have learned there, and a TARGET task. Each task comes with its
     request text and its reference
solution code. Judge the relation between the two tasks.
Rate 1-5:
procedure_same: 5 = the target is the same procedure re-run with different parameters (same
      tables, same join/filter/
aggregation skeleton, same deliverable shape); 1 = the target needs a different procedure (
     different tables, joins,
aggregation, deliverable) and the only thing shared is the fact K.
k_needed: 5 = without K a careful analyst would very likely produce a wrong answer on the
     target; 1 = K is irrelevant.
answer_copyable: true if the target answer could be copied from the source answer without
     recomputation.
regime: 'R1' (procedural reuse) if procedure_same>=4, 'R2' (data space understanding) if
     procedure_same<=2, 'mixed' otherwise.
Return JSON only: {"procedure_same":int,"k_needed":int,"answer_copyable":bool,"regime":str
     ,"reason":str(<=40 words)}
```