LEARNING THE DATA ENVIRONMENT: CONTINUAL SPECIALIZATION FOR DATA AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Data agents often answer a succession of questions about the same data sources. Although they have general analytical capabilities, they often repeat exploratory work across requests, recovering dataset-specific knowledge acquired in earlier analyses. We introduce *Continual Data Specialist* (CDS), a framework that enables a data agent to acquire reusable, environment-specific state before downstream requests are known. Using multi-table data bundles as a concrete instantiation, CDS generates analysis questions, collects tool-executed trajectories, filters them with lightweight verification, and directs later exploration toward observed failures. The resulting experience is retained through two complementary channels: a bundle-specific LoRA adapter learns patterns of analysis and tool use, while persistent memory statements make dataset-specific findings explicitly available in context. Subsequent requests reuse the same adapter and memory bank. For each request, a hypothesis module draws on both channels to propose analysis directions, which are checked against the original data through tool execution. Experiments on three data-agent benchmarks show that CDS improves insight quality over Vanilla Agent and either memory channel alone. CDS offers a route from general-purpose analysis to environment-specific expertise, turning autonomous exploration into knowledge that can support repeated use of the same data.

1 INTRODUCTION

Data agents can plan analyses, write code, and revise their work from execution feedback (Zhang et al., 2024b; Qiao et al., 2023; Guo et al., 2024; Hong et al., 2025). A general agent nevertheless begins an unfamiliar dataset with little knowledge of its identifiers, measurement conventions, or recurring patterns. Answering a focused request may require substantial preliminary work: locating files, checking keys, resolving time granularity, or determining which denominator makes a comparison meaningful. In session-based workflows, that work is often discarded with the session context.

When an organization keeps the same reports and tables available for successive requests, the understanding gained during one analysis can remain useful well beyond that session. A support team, for example, might first ask how the ticket backlog changed and later ask which categories account for slow resolution. The answers differ, but both analyses can depend on the same status definitions, timestamp conventions, and links between ticket and category tables. Retaining these discoveries could give later analysis a better starting point.

Existing frameworks and generalist training have improved agents' ability to analyze a wide range of datasets (Qiao et al., 2023; Zhang et al., 2024a; Wu et al., 2024), but knowing how to analyze data in general is different from becoming familiar with a particular collection of sources. An agent may know how to join tables or compare groups without knowing which relationships are informative in a given environment or which comparisons require special care. This familiarity develops through working with the sources themselves. Environment-specific learning thus complements general training, helping an agent apply broadly useful skills with a better understanding of the data at hand.

For this experience to improve later analyses, an agent needs a way to retain it and bring it to bear on new requests. How experience is represented and reused is therefore part of the learning problem. Interaction-based methods produce training data from target environments (Su et al., 2025;

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.

Wang et al., 2026; Qiu et al., 2026), while memory-based agents preserve feedback and guidance in context (Shinn et al., 2023; Zhang et al., 2025). Summarization and compression can make earlier interactions easier to carry forward, yet a concise record of their outcomes may leave the analytical choices implicit. We seek to retain both useful findings and experience of the decisions and tool interactions that produced them. For a new request, these should suggest where and how to investigate.

Because the sources remain available across requests, an agent can explore them proactively and prepare reusable knowledge before specific questions arrive. We introduce *Continual Data Specialist* (CDS), a framework for *bundle-specific specialization*. A *data bundle* is a fixed collection of sources that can support multiple analytical requests. Before those requests are revealed, CDS runs a single preparation phase for the bundle and constructs two memory channels: parametric memory in a LoRA adapter and contextual memory in a bank of textual statements. Both are available to the agent when answering a new request. The textual bank makes useful findings directly available, while training on analytical trajectories allows the adapter to learn from the decisions and tool interactions that produced them. All questions associated with the bundle reuse this state, while the shared base model remains frozen. A new bundle receives its own state. This organization provides a building block for continual data agents that acquire and preserve knowledge of successive environments. Each source version and its learned state remain fixed across downstream requests.

CDS has three stages (Figure 1). Autonomous question generation and Python execution first produce analytical trajectories; a lightweight filter and failure feedback control which experience is retained and where exploration proceeds. Accepted trajectories then provide three training views for LoRA and compact statements for persistent context. Finally, a hypothesis module asks the adapted agent to turn the retained information into candidate claims and executable tests for a new request. CDS checks these candidates against the original sources. Experience guides the analysis, while fresh observations determine which findings the response can support.

Experiments on InsightBench, InsightEval, and UniDataBench compare memory configurations and examine the synthesis, internalization, and application stages (Sahu et al., 2025; Zhu et al., 2026; Weng et al., 2026). Parametric and contextual memory each improve scores on InsightBench and InsightEval; combining them in CDS yields the highest scores across all three benchmarks. These gains are obtained using the base model itself to generate exploratory trajectories, training targets, and memory statements, without supervision distilled from a stronger teacher. Component studies then follow CDS's stages: quality feedback improves the synthesis configuration, supervision views affect how experience is internalized, and source-based checking turns tentative hypotheses into better-supported answers.

Our contributions are:

- **A bundle-specific experience pipeline.** We organize question generation, tool rollouts, analysis-quality filtering, and failure-directed exploration around a fixed bundle before downstream requests are known. This pipeline turns the dataset into reusable analytical experience.
- **Retaining and reusing analytical experience.** We study how agents can preserve prior experience through different supervision views and memory representations. This investigation leads to a dual-channel design combining LoRA adaptation with textual memory, and a hypothesis mechanism that draws on both to guide analysis and source-based checks for new requests.

2 Related Work

Context management in general agents. Summarization and compaction carry prior interaction state into later model calls. The OpenAI Agents SDK cookbook describes a summary-based handoff that preserves a reusable snapshot of earlier turns (Okcular, 2025); the Responses API supports compaction for continuing long-running interactions with a smaller context (OpenAI, 2026). These mechanisms address continuity as interaction history grows. Our summary handoff baseline instantiates this general pattern at trajectory boundaries. CDS instead prepares knowledge for repeated analysis of a particular source collection, combining explicit findings with adaptation from analytical trajectories.

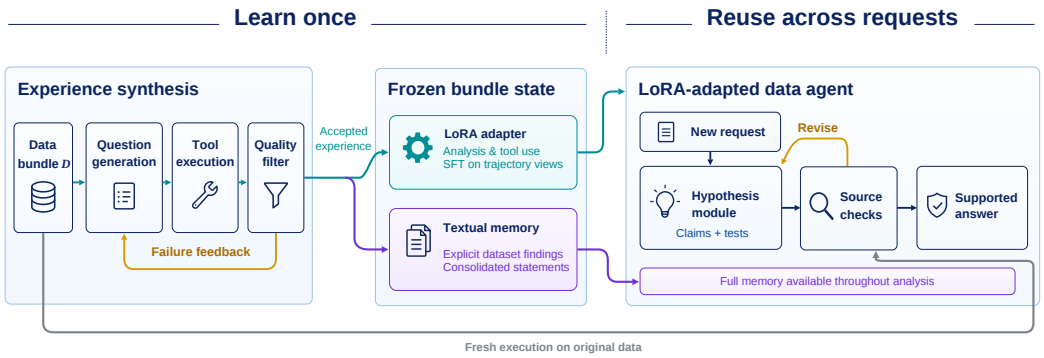


Figure 1: CDS: learn once, reuse across requests. Accepted exploration experience trains a bundle-specific adapter and supplies explicit textual findings. The adapter is loaded into the data agent, and the full memory bank remains available throughout hypothesis generation, source execution, and revision. Both retained states stay fixed across requests. Original data supplies fresh evidence for the answer.

Data agents and generalist training. Data-agent research develops both analytical workflows and general-purpose capabilities. Data-Copilot, TaskWeaver, DS-Agent, LAMBDA, and Data Interpreter develop planning, code execution, example retrieval, and workflow verification for data analysis (Zhang et al., 2024b; Qiao et al., 2023; Guo et al., 2024; Sun et al., 2026; Hong et al., 2025). AgentAda generates goal-relevant questions and selects analytical skills for tailored insights (Abaskohi et al., 2025). TableLlama, DACO, AgentTuning, and CodeAct train table, code, or agent capabilities across tasks (Zhang et al., 2024a; Wu et al., 2024; Zeng et al., 2024; Wang et al., 2024).

Learning through interaction. Interaction provides experience from which an agent can learn about its environment. TOFFEE is a close precedent for the synthesis stage: it constructs SFT and ICL trajectories for a target data environment using tree search, adaptive model selection, and prefix reuse (Wang et al., 2026). Learn-by-Interact also produces interaction data for fine-tuning or in-context use, while DSGym collects data-science trajectories (Su et al., 2025; Nie et al., 2026). AutoDiscovery explores datasets through open-ended hypotheses and experimental feedback (Agarwal et al., 2025). Voyager combines autonomous curricula with executable feedback, and Agent0 couples a curriculum proposer with a tool-using executor (Wang et al., 2023; Xia et al., 2025). CDS focuses on the long-term use of a particular data bundle: retaining dataset knowledge acquired through exploration in parametric and textual memory for subsequent requests, with the aim of reducing repeated exploration.

Procedural and textual memory. Experience can also be retained as reusable skills and written feedback. DataCOPE discovers verifier-guided analytical skills while keeping backbone parameters fixed (Qiu et al., 2026). EvoDS trains skill acquisition and context management through SFT and agentic reinforcement learning (Yang et al., 2026). Reflexion stores linguistic feedback across trials, and ACE curates a context playbook from execution feedback (Shinn et al., 2023; Zhang et al., 2025). CDS pairs an explicit bank of analytical findings with a bundle-specific adapter, exposing both to the hypothesis module during each request.

Source-specific and continual adaptation. Another approach is to adapt a model to a particular source of information. SEAL generates model updates for incorporating new knowledge (Zweiger et al., 2025). Cartridges learns a corpus-specific KV cache through synthetic self-study; Deep Context Distillation trains document-specific LoRA modules that can complement retrieval (Eyuboglu et al., 2025; Caccia et al., 2025). Test-time learning likewise uses target-domain inputs for adaptation (Sun et al., 2020; Hardt & Sun, 2024; Hu et al., 2025), while parameter-isolation approaches allocate separate capacity to preserve earlier task parameters (Rusu et al., 2016; De Lange et al., 2022). CDS combines source-conditioned learning with tool-executed analytical experience: one adapter

and textual bank retain what exploration produced for a bundle, and its later requests continue to access the original data.

3 BUNDLE-SPECIFIC EXPERIENCE SYNTHESIS

We study tool-using data analysis, with multi-table datasets as the main setting. Given a collection of data sources and a natural-language request, an agent plans an analysis, executes code, and interprets the resulting observations. For example, sales records and a product table can be used to identify which products sell well or how sales change over time. The same sources often support many requests, motivating an agent to retain what it learns about the data for later analysis.

We call this collection a data bundle D. CDS runs one preparation phase per bundle, including iterative exploration and adapter training, to learn a reusable state $S = (\phi^\star, M)$ consisting of LoRA parameters $\phi^\star$ and a textual memory bank M. Subsequent requests over the bundle share this state. For clarity, the notation below describes one bundle, omitting bundle-specific subscripts. Specialization uses only its source data and permitted metadata, before downstream questions and reference answers are revealed.

Synthesis turns access to a dataset into experience of analyzing it. Questions guide tool-based exploration, which produces both discoveries about the bundle and examples of how they were obtained. Figure 1 shows how this experience becomes retained state for later requests.

3.1 QUESTION GENERATION FROM THE BUNDLE

We begin with a broad view of the bundle, then use it to propose concrete analyses. A profile P contains source schemas, column types, basic statistics, representative values, and permitted metadata. Tables and files remain grouped under their bundle so that a question can concern either an individual source or a relationship between sources. This profile grounds question generation in the available data without requiring knowledge of future requests.

From this profile, the initial question pool Q_0 combines three sources and removes duplicates:

$$Q_0 = \mathrm{Dedup}(Q_{\mathrm{tmpl}} \cup Q_{\mathrm{stat}} \cup Q_{\mathrm{LM}}). \tag{1}$$

The subscripts identify the source of each question set. Template-based questions (tmpl) cover common operations such as distributions, subgroup comparisons, and temporal trends. Statistics-driven questions (stat) use the profile's variable types and summary statistics to instantiate analyses suited to the bundle. Language-model-generated questions (LM) use the profile's descriptions and observed values to propose additional dataset-specific directions. The union combines these sets, and Dedup removes repeated questions to produce a single pool. The resulting questions initiate exploration; execution and quality feedback subsequently determine which trajectories are retained.

3.2 EXECUTED TRAJECTORIES AND QUALITY FEEDBACK

We execute the generated questions, retain trajectories that pass screening, and use rejected cases to guide further question generation.

Tool-based rollout. An exploration model answers each sampled question through a Python tool, using the reasoning-and-acting interface of Yao et al. (2023). A trajectory records the question, assistant messages, executed programs, tool observations, and final response. The model can change its analysis after seeing an observation or an execution error. These traces supervise both findings and the actions used to obtain them. The exploration model supplies preparation-time experience; the downstream model uses the bundle-specific LoRA parameters. Model settings for the experiments are given in Section 5.1.

Verification-aware filtering. Quality feedback considers both the execution process and how well the resulting analysis addresses the starting question. Before the acceptance decision, a completed rollout is used to draft a detailed analysis, its final findings, and a compact memory statement. Structural checks require a complete interaction, at least one successful execution, and at most two failed

tool calls. A language model serves as the judge, receiving the question and detailed analysis to assess relevance and coverage. Its judgments can be inaccurate, and passing this check does not guarantee that the analysis is factually correct. We therefore call this screening step *semi-verification*. Accepted trajectories retain their drafted supervision; rejected ones receive a coarse failure label.

Failure-driven curriculum. An initial question pool may cover familiar analyses while leaving harder questions and parts of the data bundle insufficiently explored. Failure-driven generation directs further collection toward analyses the model has not yet completed reliably, creating additional opportunities to learn about the corresponding data and relationships. Starting from Q_0, each synthesis round uses execution and filtering outcomes to generate supplementary questions:

$$Q_r \xrightarrow{\text{execute}} T_r \xrightarrow{\text{filter}} F_r \xrightarrow{\text{generate and deduplicate}} Q_{r+1}. \tag{2}$$

Here, Q_r is the question batch in round r, T_r its collected trajectories, and F_r the failure feedback from filtering. Accepted trajectories are retained separately and accumulate across rounds. Feedback groups rejected cases by analysis type and reason: missing execution, repeated tool errors, irrelevance, or incomplete coverage. The generator uses this feedback together with the fixed profile P to target weak operations. The loop stops at the synthesis budget; only synthesis-time outcomes guide the curriculum.

Training views and memory statements. CDS represents each accepted analysis through three supervision views. The *original trajectory* preserves the assistant messages, tool calls, and execution results. The *detailed analysis* describes in natural language how the agent analyzed the question and arrived at its findings. The *final findings* state the conclusions reached through that analysis. Examples that pass formatting checks enter the training corpus C. In parallel, compact memory statements retain selected findings in the textual bank M for inference context. These statements are separate from the three LoRA supervision views. Appendix A.1 gives the training input–target formats.

4 Memory Internalization and Application

The accepted experience has two destinations. It trains an adapter, and its statements are consolidated into a fixed textual bank. Figure 1 shows these paths and the subsequent hypothesis-and-execution loop. Both channels are associated with the bundle and remain unchanged while downstream requests are answered.

4.1 Parametric and Textual Memory

LoRA adaptation. Exploration reveals not only facts about a bundle, but also ways of working with it: which tables to consult, how to structure a comparison, when to call a tool, and how to proceed after an observation. Training on these interactions gives the model examples of analytical behavior in the target environment. CDS uses LoRA (Hu et al., 2022) to retain this experience through low-rank updates to a frozen base model. This choice fits bundle-specific learning: the shared backbone supplies general reasoning and coding capabilities, while a smaller set of trainable parameters provides a separate adaptation for each bundle. The resulting adapters can be stored and selected independently without maintaining a full model copy for every environment.

Let p_θ be the downstream base model. CDS freezes θ and trains the adapter parameters ϕ on corpus C:

$$\phi^\star = \arg\min_\phi \sum_{z \in C} \mathcal{L}_{\text{assistant}}(z; \theta, \phi). \tag{3}$$

Here, z is a training example and $\mathcal{L}_{\text{assistant}}$ sums next-token negative log likelihoods over its assistant tokens; tool observations provide conditioning context. Original trajectories supervise tool calls and continuation from observations. Detailed analyses describe the reasoning behind the findings, and final findings provide the resulting conclusions. Their contributions follow the number of target tokens under the shared objective. The resulting checkpoint is registered once for that bundle and loaded for each of its downstream requests.

Persistent textual memory. The textual channel expresses selected discoveries as statements CDS can read directly. Before requests are revealed, a query-independent consolidation step organizes the collected statements into the fixed bank M. A statement might record that a product category had declining sales during an explored period, together with the scope of that observation. Such a record makes a previous finding available without keeping its entire conversation in the prompt. The entire bank is included in every downstream context.

The channels preserve complementary aspects of the same experience: textual statements make findings explicit, while LoRA learns from the broader pattern of actions, observations, and analysis.

4.2 Hypotheses Followed by Source Checks

One checkpoint, multiple requests. CDS stores the state S under a bundle-level key and maps all associated requests to it. Each request starts a fresh analysis session with the same adapter weights and the entire retained textual bank. The source version, adapter, and bank remain fixed across these sessions. A bundle with K requests therefore has one synthesis-and-training stage followed by K inference episodes. Tool observations and provisional hypotheses belong to the current episode; the retained state carries the reusable experience.

Hypothesis-based memory elicitation. For a downstream request q, CDS loads $\phi^\star$ and supplies M as context. The adapted model then proposes a set H of candidate claims paired with executable analysis plans and expected evidence:

$$H = \mathrm{Hyp}(q, M; p_{\theta,\phi^\star}). \tag{4}$$

The module elicits bundle information as testable analysis leads, using the adapted model together with the textual bank. For example, a remembered pattern may suggest that a product category has a higher return rate; the plan then specifies the relevant time interval, category grouping, and denominator. Each candidate defines an analysis direction to test against source observations.

The executable plan specifies the scope and observations needed to assess a claim. Each check follows the scope and conditions specified by the current request. The full textual bank remains visible throughout execution and revision.

Evidence refinement. CDS executes candidate plans over the original sources and compares the observations with the proposed claims. Supporting evidence can retain or refine a claim, contradictions lead to revision or rejection, and insufficient evidence leaves it unresolved. CDS continues exploration when further analysis is needed. Its response reports supported findings and qualifies unresolved questions.

5 Experiments

We evaluate CDS and then follow its three stages: experience synthesis, internalization, and request-time application. The experiments address four questions: how CDS compares with Vanilla Agent and alternative memory configurations; how synthesis feedback affects the retained experience; which supervision views help internalization; and why memory-informed hypotheses need source-based checking.

5.1 Evaluation Setup

Benchmarks and data units. We evaluate on 100 tasks from each of InsightBench, InsightEval, and UniDataBench (Sahu et al., 2025; Zhu et al., 2026; Weng et al., 2026). Each task specifies an analytical request and its source files, whose complete collection defines the data bundle. Multiple tasks can concern the same bundle. InsightBench and InsightEval provide distinct task/reference protocols over largely shared source tables. Appendix B details source-bundle structure and request sharing.

Model and memory configurations. The experiments use Qwen3.6-27B for exploration, target drafting, filtering, and downstream analysis. CDS combines bundle-specific LoRA, persistent memory statements, hypothesis generation, and source-based refinement. Dataset-specific LoRA and

CDS share the same per-bundle checkpoint, trained on three views: original trajectory, detailed analysis, and final findings. The LoRA-only configuration omits the textual bank. Requests for a bundle reuse its checkpoint; textual-memory conditions supply the full bank. CDS rows across tables repeat the same evaluations. Appendix A gives the training and implementation settings.

Summary handoff baseline. Summary handoff ICL maintains context across successive trajectories: after answering each question and completing its trajectory, the agent receives a summary prompt, and the resulting summary is placed at the beginning of the next trajectory. This baseline passes experience forward through post-trajectory summaries, whereas CDS's retained statement bank is constructed before downstream requests and remains fixed across them. The baseline is implemented with the same Qwen3.6-27B backbone, following the summary-based context-management pattern used in general agent workflows (Okcular, 2025). Appendix A records training and supervision details; Appendix A.2 specifies the per-question scoring implementation.

Metrics. GPT-4o provides G-Eval-style assessment against the benchmark reference analyses (Liu et al., 2023). All methods within each benchmark are evaluated on the same 100 questions. We report the mean question-level score, giving each question equal weight, including when several questions share a data bundle. Grounded Claims is the mean per-question proportion of claims judged supported by evidence. DeepSeek V4 Pro assesses support with access to tool execution results and the original source data; question-level proportions are averaged with equal weight.

5.2 Memory Configurations

Table 1: Mean G-Eval-style scores for Qwen3.6-27B. Every method uses the same 100-question set within each benchmark. LoRA checkpoints are shared by questions belonging to the same bundle. Bold marks the largest score in each column.

Configuration	InsightBench	InsightEval	UniDataBench
Vanilla Agent	0.5583	0.6177	0.7164
Summary handoff ICL	0.5561	0.6280	0.7141
Memory statement ICL	0.6238	0.6485	0.7372
Dataset-specific LoRA	0.6197	0.6331	0.7152
CDS	**0.6552**	**0.6653**	**0.7648**

Preparing dataset-specific findings gives larger gains than passing session summaries (Table 1). Summary handoff stays close to Vanilla Agent, while the statement bank improves scores on all three benchmarks. The bank exposes discoveries collected before evaluation; handoff passes summaries of completed requests. The results favor preparing reusable findings through bundle exploration, although these strategies also differ in when and how they acquire experience.

Parametric memory is most useful alongside textual findings. LoRA alone improves InsightBench and InsightEval, with little change on UniDataBench. CDS nevertheless exceeds the strongest single channel, Memory statement ICL, by 0.0168–0.0314 across the benchmarks. This includes UniDataBench, where the adapter alone contributes little, suggesting that the benefit of adaptation depends on how it is combined with explicit knowledge.

Across all three benchmarks, CDS's dual-channel experience-reuse strategy improves on either memory channel alone and yields higher insight scores than direct summary handoff.

5.3 Experience Synthesis

Table 2 compares three ways of collecting synthetic experience. The first retains trajectories using execution checks alone. The second adds a model-based check of the detailed analysis for relevance and coverage. CDS also uses failure feedback to generate supplementary questions. Each row is evaluated through the resulting agent's downstream answers on InsightBench.

Table 2: Feedback components in experience synthesis. All variants use execution checks; checkmarks indicate additional components. Scores measure downstream insight quality on InsightBench.

Synthesis setting	Analysis-quality check	Failure-guided questions	Score ↑
Execution checks only	×	×	0.6281
+ Analysis-quality check	✓	×	0.6431
CDS	✓	✓	**0.6552**

Analysis-quality screening adds 0.0150 over execution checks alone. A successful tool call establishes that a computation ran, but its output may address only part of the question or support an irrelevant conclusion. Checking the written analysis helps select experience that connects execution results to the requested insight. The higher downstream score is consistent with the usefulness of this additional selection criterion.

Failure-guided questions add a further 0.0121. Filtering improves selection from collected trajectories; it can still leave little usable experience for analyses the model repeatedly struggles with. Failure feedback gives those operations further attention through supplementary questions and rollouts. The combined configuration gains 0.0271 over execution-only filtering, supporting feedback at both selection and question-generation stages. These are pipeline-level comparisons: the amount and composition of retained experience also change.

5.4 Multi-View Internalization

We examine how different representations of the same analysis affect internalization. The first three rows of Table 3 start with the original trajectory, then add detailed analysis and final findings. The three-view configuration corresponds to Dataset-specific LoRA in Table 1. The CDS row uses the same adapter and additionally supplies textual memory.

Table 3: Cumulative LoRA supervision views. CDS uses the same three-view checkpoint as the preceding row, with textual memory at inference, and repeats the CDS result in Table 1.

	G-Eval-style score ↑	
Internalization setting	InsightBench	InsightEval
Original trajectory	0.5632	0.6105
+ Detailed analysis	0.6102	0.6395
+ Final findings	0.6197	0.6331
CDS	**0.6552**	**0.6653**

The most consistent improvement comes from explaining the discovery process in natural language. Adding detailed analysis improves InsightBench and InsightEval by 0.0470 and 0.0290. This view makes explicit how tool operations and observations lead to findings. The final-findings view has a smaller, benchmark-dependent effect: it further improves InsightBench but slightly lowers InsightEval. Supervision content is therefore worth examining alongside the amount of supervised text, which also changes in this comparison.

The textual bank also remains useful after three-view training. With the adapter unchanged, supplying the bank improves both benchmarks. Retaining an explicit record of discoveries thus complements the learned state at inference.

These results suggest that original trajectories alone leave room for improvement: natural-language accounts of the discovery process and its conclusions may help generalization to new requests. We leave a systematic study of these supervision formats to future work.

Table 4: Application comparison on InsightBench. Vanilla Agent uses the base model without LoRA or retained textual memory and retains standard tool access. Refinement denotes explicit hypothesis checking. Insight score is the G-Eval-style aggregate; Grounded Claims averages each question's proportion of claims judged supported by tool results or source data.

Application setting	Hypothesis	Refinement	Insight score ↑	Grounded Claims ↑
Vanilla Agent	×	×	0.5583	**0.91**
Hypothesis only	✓	×	0.4972	0.24
CDS	✓	✓	**0.6552**	0.86

5.5 Hypotheses and Source Verification

Hypothesis-only inference exposes the risk of carrying plausible candidates directly into an answer (Table 4). It scores below Vanilla Agent, with a Grounded Claims rate of 24%. Source checking raises support to 86% and improves reference agreement. Such checks matter when a new request changes the subset or aggregation to which a remembered finding applies.

The comparison with Vanilla Agent reveals a quality trade-off: CDS achieves higher reference agreement (0.6552 versus 0.5583), while Vanilla Agent has the higher supported-claim proportion (91% versus 86%). Source checks substantially improve hypothesis-led answers, although a gap in evidence support remains relative to the baseline.

6 Conclusion

We presented CDS, a framework for turning autonomous data exploration into reusable environment-specific knowledge. Tool-executed experience trains a bundle-specific adapter and supplies explicit findings, while a hypothesis-and-checking procedure connects both to new requests. Experiments support joint parameter and textual memory and show that supervision construction and source-based refinement matter for answer quality. CDS takes persistent data environments as a setting in which agents can acquire lasting analytical expertise alongside their general capabilities.

7 Limitations

Training overhead. Preparing a bundle-specific adapter is time-consuming: CDS collects exploratory trajectories and trains a separate LoRA for each data bundle. Its practical value therefore depends on repeated use, with many follow-up questions sharing the preparation cost. For datasets queried only a few times, the additional training time may outweigh the benefits of improved answers. Measuring this trade-off across request volumes is an important direction for further evaluation.

Evaluation coverage. Our evaluation covers InsightBench, InsightEval, and UniDataBench. The evidence is therefore concentrated on these three data-agent benchmarks. Broader evaluation on general-purpose data analysis benchmarks, spanning more data domains and analytical workflows, is needed to establish how well CDS generalizes beyond the current setting.

AI use statement

Generative AI tools assisted with literature review, mathematical formulation, language polishing, and figure preparation. CDS uses language models to generate exploration questions, trajectories, training targets, and memory statements, as well as for screening and evaluation, as described in the paper. The authors are responsible for the final content and for validating the methods, citations, results, and AI-assisted contributions.

REFERENCES

Amirhossein Abaskohi, Amrutha Varshini Ramesh, Shailesh Nanisetty, Chirag Goel, David Vazquez, Christopher Pal, Spandana Gella, Giuseppe Carenini, and Issam H. Laradji. AgentAda: Skill-adaptive data analytics for tailored insight discovery, 2025. URL https://arxiv.org/abs/2504.07421.

Dhruv Agarwal, Bodhisattwa Prasad Majumder, Reece Adamson, Megha Chakravorty, Satvika Reddy Gavireddy, Aditya Parashar, Harshit Surana, Bhavana Dalvi Mishra, Andrew McCallum, Ashish Sabharwal, and Peter Clark. AutoDiscovery: Open-ended scientific discovery via bayesian surprise, 2025. URL https://arxiv.org/abs/2507.00310.

Lucas Caccia, Alan Ansell, Edoardo Ponti, Ivan Vulić, and Alessandro Sordoni. Training plug-and-play knowledge modules with deep context distillation. In *Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=ghyyHZYORi.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3366–3385, 2022. doi: 10.1109/TPAMI.2021.3057446. URL https://doi.org/10.1109/TPAMI.2021.3057446.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Will Tennien, Atri Rudra, James Zou, Azalia Mirhoseini, and Christopher Ré. Cartridges: Lightweight and general-purpose long context representations via self-study, 2025. URL https://arxiv.org/abs/2506.06266.

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. DS-Agent: Automated data science by empowering large language models with case-based reasoning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 16813–16848. PMLR, 2024. URL https://proceedings.mlr.press/v235/guo24b.html.

Moritz Hardt and Yu Sun. Test-time training on nearest neighbors for large language models. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/f02f1185b97518ab5bd7ebde466992d3-Abstract-Conference.html.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Robert Tang, Xiangtao Lu, Xiawu Zheng, Xinbing Liang, Yaying Fei, Yuheng Cheng, Yongxin Ni, Zhibin Gou, Zongze Xu, Yuyu Luo, and Chenglin Wu. Data Interpreter: An LLM agent for data science. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 19796–19821. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.1016. URL https://aclanthology.org/2025.findings-acl.1016/.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2106.09685.

Jinwu Hu, Zitian Zhang, Guohao Chen, Xutao Wen, Chao Shuai, Wei Luo, Bin Xiao, Yuanqing Li, and Mingkui Tan. Test-time learning for large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 24823–24849. PMLR, 2025. URL https://proceedings.mlr.press/v267/hu25z.html.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2511–2522. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.153. URL https://aclanthology.org/2023.emnlp-main.153/.

Fan Nie, Junlin Wang, Harper Hua, Federico Bianchi, Yongchan Kwon, Zhenting Qi, Owen Queen, Shang Zhu, and James Zou. DSGym: A holistic framework for evaluating and training data science agents, 2026. URL https://arxiv.org/abs/2601.16344.

Emre Okcular. Context engineering—short-term memory management with sessions. OpenAI Cookbook, 2025. URL https://developers.openai.com/cookbook/examples/agents_sdk/session_memory. Published September 9, 2025; accessed September 26, 2026.

OpenAI. Compaction. OpenAI API Documentation, 2026. URL https://developers.openai.com/api/docs/guides/compaction. Accessed September 26, 2026.

Bo Qiao, Liqun Li, Xu Zhang, Shilin He, Yu Kang, Chaoyun Zhang, Fangkai Yang, Hang Dong, Jue Zhang, Lu Wang, Minghua Ma, Pu Zhao, Si Qin, Xiaoting Qin, Chao Du, Yong Xu, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. TaskWeaver: A code-first agent framework, 2023. URL https://arxiv.org/abs/2311.17541.

Zhisong Qiu, Kangqi Song, Shengwei Tang, Shuofei Qiao, Lei Liang, Huajun Chen, and Shumin Deng. Unsupervised skill discovery for agentic data analysis, 2026. URL https://arxiv.org/abs/2606.06416.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks, 2016. URL https://arxiv.org/abs/1606.04671.

Gaurav Sahu, Abhay Puri, Juan A. Rodriguez, Amirhossein Abaskohi, Mohammad Chegini, Alexandre Drouin, Perouz Taslakian, Valentina Zantedeschi, Alexandre Lacoste, David Vazquez, Nicolas Chapados, Christopher Pal, Sai Rajeswar Mudumba, and Issam Hadj Laradji. InsightBench: Evaluating business analytics agents through multi-step insight generation. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=ZGqd0cbBvm.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Hongjin Su, Ruoxi Sun, Jinsung Yoon, Pengcheng Yin, Tao Yu, and Sercan O. Arik. Learn-by-Interact: A data-centric framework for self-adaptive agents in realistic environments. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=3UKOzGWCVY.

Maojun Sun, Ruijian Han, Binyan Jiang, Houduo Qi, Defeng Sun, Yancheng Yuan, and Jian Huang. LAMBDA: A large model based data agent. *Journal of the American Statistical Association*, 121(553):1–13, 2026. doi: 10.1080/01621459.2025.2510000. URL https://doi.org/10.1080/01621459.2025.2510000.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9229–9248. PMLR, 2020. URL https://proceedings.mlr.press/v119/sun20b.html.

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, 2023. URL https://arxiv.org/abs/2305.16291.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 50208–50232. PMLR, 2024. URL https://proceedings.mlr.press/v235/wang24h.html.

Ziting Wang, Yin Li, Zuhao Yang, Xiuchang Li, Jiale Bai, and Gao Cong. Demonstrating TOFFEE: A learned system for synthesizing data agent trajectories at scale, 2026. URL https://arxiv.org/abs/2607.06233. VLDB 2026 Demonstration.

Han Weng, Zhou Liu, Yuanfeng Song, Xiaoming Yin, Xing Chen, and Wentao Zhang. UniDataBench: Evaluating data analytics agents across structured and unstructured data. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 33755–33780. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1556. URL https://aclanthology.org/2026.acl-long.1556/.

Xueqing Wu, Rui Zheng, Jingzhen Sha, Te-Lin Wu, Hanyu Zhou, Mohan Tang, Kai-Wei Chang, Nanyun Peng, and Haoran Huang. DACO: Towards application-driven and comprehensive data analysis via code generation. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, volume 37, 2024. doi: 10.52202/079017-2878. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/a4cb1444fb05839d0113d2773da88a49-Abstract-Datasets_and_Benchmarks_Track.html.

Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning, 2025. URL https://arxiv.org/abs/2511.16043.

Zherui Yang, Fan Liu, Yansong Ning, and Hao Liu. EvoDS: Self-evolving autonomous data science agent with skill learning and context management. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2026. doi: 10.1145/3770855.3818002. URL https://arxiv.org/abs/2606.03841.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3053–3077. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.181. URL https://aclanthology.org/2024.findings-acl.181/.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models, 2025. URL https://arxiv.org/abs/2510.04618.

Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. TableLlama: Towards open large generalist models for tables. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 6024–6044. Association for Computational Linguistics, 2024a. doi: 10.18653/v1/2024.naacl-long.335. URL https://aclanthology.org/2024.naacl-long.335/.

Wenqi Zhang, Yongliang Shen, Weiming Lu, and Yueting Zhuang. Data-Copilot: Bridging billions of data and humans with autonomous workflow. In *ICLR 2024 Workshop on Large Language Model Agents*, 2024b. URL https://openreview.net/forum?id=iUEBnQyfWE.

Zhenghao Zhu, Yuanfeng Song, Xing Chen, Chengzhong Liu, Yakun Cui, Caleb Chen Cao, Sirui Han, and Yike Guo. InsightEval: An expert-curated benchmark for assessing insight discovery in LLM-driven data agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 34632–34656. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1729. URL https://aclanthology.org/2026.findings-acl.1729/.

Adam Zweiger, Jyo Pari, Han Guo, Yoon Kim, and Pulkit Agrawal. Self-adapting language models. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/6b41e04c41726e2a60e456d0a2b961ab-Abstract-Conference.html.

A Implementation Details

Table 5 lists key model, training, and generation parameters. Training hyperparameters and the validation configuration below are documented in the run records. Synthesis budgets and generation temperatures are code defaults.

Table 5: Key parameters for the Qwen3.6-27B memory and component studies.

Component	Setting
Backbone	Qwen3.6-27B
LoRA	Rank 16; alpha 32; all linear layers
Optimization	3 epochs; batch size 32; learning rate 3×10^{-4}
Schedule	Cosine decay; 3% warmup; weight decay 0.1
Training hardware	4 NVIDIA H20 GPUs; bfloat16
Synthesis budget	Target: 300 successful trajectories/bundle; up to 3,000 attempts
Sampling	Query temperature 0.95; rollout temperatures 0.3–0.8
Drafting / screening	Temperatures 0.2 / 0
Inference	Temperature 0
Reference judge	GPT-4o; temperature 0

Validation monitoring. Validation monitoring uses the training data itself, with no held-out split.

A.1 Training-View Construction

Original trajectories are trained in their existing conversation format. For both detailed analysis and final findings, the input is the question and the target is the corresponding text derived from the trajectory. Memory statements are retained separately for inference-time context.

Each view is formatted as a separate training example. The supervised objective masks non-assistant tokens and introduces no separate view-specific weighting.

A.2 Scoring Details

Per-question judge score. The scoring procedure supplies the generated answer and benchmark reference to a GPT-4o judge. It asks for a 1–10 closeness rating, with 10 indicating agreement with the reference, formatted as `<rating>r</rating>`. At the rating-token position, let $\mathcal{T}_i$ contain the five returned candidate tokens and let ℓ_{ij} be their log probabilities. The normalized score is

$$s_i = \frac{1}{10} \sum_{j \in \mathcal{T}_i} \frac{\exp(\ell_{ij})}{\sum_{k \in \mathcal{T}_i} \exp(\ell_{ik})} \, v(t_{ij}), \tag{5}$$

where $v(t)$ is the integer value of a digit token and zero for a nonnumeric token. The normalization covers all returned candidates, including nonnumeric ones. Scores are averaged over the same 100 questions for every method, as specified in Section 5.1.

Grounded Claims. DeepSeek V4 Pro evaluates whether the reported claims are supported by evidence, with access to tool execution results and the original source data. For each question, the support rate is the number of claims judged supported divided by the number of evaluated claims. Grounded Claims reports the arithmetic mean of these question-level rates on a scale from 0 to 1, giving each question equal weight regardless of its number of claims.

B Source-Bundle Grouping

Grouping rule. We group the first 100 tasks in each evaluated InsightBench and InsightEval subset by their complete source files, including primary and auxiliary tables. A bundle is identified by

the sorted set of its files' SHA-256 content hashes, independently of filenames. Distinct goals are counted after whitespace normalization. Table 6 summarizes the resulting bundles and multi-goal requests.

Table 6: Source structure of the two evaluated 100-task subsets. A multi-goal bundle contains at least two distinct goal strings after whitespace normalization. Bundle identity includes auxiliary sources.

Benchmark	Tasks	Source bundles	Multi-goal bundles	Tasks in those bundles
InsightBench	100	51	2	33
InsightEval	100	47	18	71

Request multiplicity. Table 7 gives the number of task records per bundle. InsightBench includes repeated goals over the same sources, yielding 76 distinct bundle–goal pairs; all 100 pairs in InsightEval are distinct. Each bundle's requests share one checkpoint and start separate analysis sessions. Benchmark scores retain the original task records and their references.

Table 7: Source-bundle multiplicities in the evaluated subsets. Each cell counts bundles with the specified number of task records.

Benchmark	1 task	2 tasks	4 tasks	29 tasks	33 tasks
InsightBench	33	15	2	1	0
InsightEval	29	15	2	0	1

C EXAMPLE SYNTHESIZED TRAJECTORY

We illustrate the synthesis stage with an archived trajectory over `flag-1.csv`, a source containing 500 ticket records. Code paths, comments, and printed headers are shortened; observations are typeset for readability. The displayed counts were checked against the source data.

User query. "What is the cross-tabulation between 'state' and 'closed_by'? Create a contingency table."

Tool call 1: compute and display the cross-tabulation. The agent invokes `run_python` with the following normalized code:

```
import pandas as pd
df = pd.read_csv('flag-1.csv')
ct = pd.crosstab(df['state'], df['closed_by'])
print(ct)
print(ct.sum())
```

The call succeeds, but the default display elides the middle column with `...` and reports `[2 rows x 5 columns]`. Column totals (95, 98, 96, 92, 119) remain visible. The agent reprints the result to expose every cell.

Tool call 2: obtain the complete display. The second `run_python` call reloads the same source, recomputes the same cross-tabulation, and changes only the display settings and printed totals:

```
import pandas as pd
df = pd.read_csv('flag-1.csv')
ct = pd.crosstab(df['state'], df['closed_by'])
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
print(ct)
print(ct.sum())
print(ct.sum(axis=1))
```

This call also succeeds. The full observation, with separately printed margins combined into one table, is:

State	Beth Anglin	Charlie Whitherspoon	Fred Luddy	Howard Johnson	Luke Wilson	Total
Closed	45	51	55	46	62	259
Resolved	50	47	41	46	57	241
Total	95	98	96	92	119	500

Final answer (condensed). `INSIGHT:` Luke Wilson appears most often in `closed_by`: 119 records, comprising 62 Closed and 57 Resolved tickets. Howard Johnson has an equal number in each state (46 Closed and 46 Resolved). Overall, there are 259 Closed and 241 Resolved tickets.

Stored memory statement (verbatim). "The dataset 'flag-1.csv' contains ticket data with 'state' (Closed/Resolved) and 'closed_by' columns, where Luke Wilson is the most active closer and Howard Johnson exhibits a balanced closure pattern."

Filtering outcome. The trajectory passed the relevance and completeness checks, and its tool calls completed without execution errors. The stored phrase *most active closer* compresses a record count into a broader description; *balanced closure pattern* refers only to the equal state counts here. A passing relevance/completeness gate therefore does not establish the precision of every phrase retained in memory.