Toward Self-Evolving Data Agents for Autonomous Data Analysis

Junhao Zhu
Zhejiang University
Hangzhou, China
zhujunhao@zju.edu.cn

Lu Chen
Zhejiang University
Hangzhou, China
luchen@zju.edu.cn

ABSTRACT

Autonomous LLM agents have shown strong promise in coding, where compilers, tests, and runtime errors provide clear feedback for iterative improvement. Data analysis poses a harder setting: tasks are heterogeneous, correct answers are often unknown before analysis, and agents must ground every step in task-local data while avoiding hallucinated workflows. This paper studies whether a data-agent harness can evolve itself through a Meta-Harness workflow. Starting from a ReAct-style baseline, the workflow repeatedly runs heterogeneous data-analysis tasks, evaluates outputs, inspects execution traces and failure modes, proposes harness-level edits, and reruns the benchmark. The resulting harness, midastouch, coordinates multiple sub-agents for knowledge injection, planning, bounded tool-using execution, verification, and answer normalization, while integrating tools for data profiling, document-to-table extraction, and other data-centric operations. Experiments show that autonomous harness evolution consistently improves data-analysis performance from 0.356 to 0.717, demonstrating the effectiveness of self-evolution for data agents. Moreover, comparisons against stronger model and coding-agent competitors further indicate that both domain-specific agent runtime structure and foundation-model strength matter for autonomous data analysis.

VLDB Workshop Reference Format:
Junhao Zhu and Lu Chen. Toward Self-Evolving Data Agents for Autonomous Data Analysis. VLDB 2026 Workshop: ADS 2026: The Joint Workshop on Agentic Data Systems and Data-Centric AI (The 1st ADS & 3rd DATAI).

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/JunHao-Zhu/midastouch.

1 INTRODUCTION

Large language models (LLMs) have enabled autonomous (multi-)agent systems that can process complex tasks from fuzzy natural-language (NL) specifications, significantly reshaping human workflows. This success has been especially visible in coding, where increasingly mature agentic coding products such as Claude Code and Codex have emerged. From the perspective of a data practitioner, a natural question is: *Could such success be transferred to the data domain?* Can an agent serve as an experienced data analyst who can explore, understand, and manipulate data, and mine insights from it?

However, data analytics and coding provide very different environments for LLM agents. (1) Unclear passing signals. A coding agent can receive clear signals from its environment. For example, the underlying compiler may report errors when it compiles generated code, or a coding agent can write unit tests to verify whether the code behaves as expected. In data analysis, however, verifiable passing signals are often unavailable. On the one hand, a golden answer is unknown before the analysis is complete; on the other hand, data analysis is case-by-case and data-dependent, so there is no standard workflow for evaluation. (2) Constrained reasoning. To solve a data-analysis task, an agent must act and reason while remaining grounded in the given data to avoid LLM hallucination. Data understanding therefore becomes a first-class problem, and intermediate results become more important. As shown by a recent benchmarking study [11], even the best LLMs achieve low scores on data-analytics tasks.

Multiple studies have attempted to close this gap from either the LLM side or the agent-harness side: (1) LLM-wise autonomous data analytics [4, 8, 15]. These works aim to enhance base-model capabilities for data processing by fine-tuning models on real or synthetic training data with specific training regimes. (2) Agent-wise autonomous data analytics [3, 7]. Another line of work focuses on agent-harness design to ensure that an LLM can execute data-analysis tasks correctly and precisely. Both lines of work rely on substantial manual engineering effort.

Motivated by the success of self-optimization methods in coding and research agents, from textual prompt self-optimization [2, 14] to agent-harness self-optimization [6, 10], we ask: *Can a data-agent harness evolve itself?* In this paper, we study this question through a Meta-Harness workflow for autonomous data-agent evolution. The workflow starts from a ReAct-style baseline harness, runs it on a development set of heterogeneous data-analysis tasks, evaluates the produced answers, inspects execution traces and errors, proposes harness-level improvements, edits the codebase, and reruns the benchmark. In this way, failures are not merely post-hoc observations; they become design signals that are converted into new harness mechanisms.

The evolved harness, named midastouch, separates data analysis into a lead agent and several specialist subagents. Before acting, the harness grounds the agent with task-local knowledge and data profiles. A planner then produces a high-level strategy, while the lead agent executes tool calls in a bounded ReAct loop. Candidate answers are audited by a verifier and normalized into a safe final table before being written for scoring. This architecture is designed around recurring failure modes observed during evolution: missing side sources, premature execution, context overflow, inconsistent

Proceedings of the VLDB Endowment. ISSN 2150-8097.

tool outputs, unsafe answer shaping, weak document extraction, and verifier-induced regressions.

We evaluate data-agent self-evolution on a set of heterogeneous data-analysis tasks. Our experiments reveal two key findings. First, harness evolution is highly effective: the final harness more than doubles the evaluation score of the initial ReAct-style baseline, showing that many data-agent failures can be repaired by improving the runtime architecture rather than only changing the model. Second, harness design is as important as raw model strength. The evolved data-specific harness outperforms a general-purpose SOTA coding agent and remains competitive across backend models, suggesting that heterogeneous data analysis requires domain-specific runtime structure in addition to strong language-model reasoning. Finally, we discuss the limitations: although effective, data-agent harness self-evolution still suffers from overfitting and local optima.

2 PROBLEM DEFINITION

This section formalizes heterogeneous data-analysis tasks and the optimization problem for data-agent harnesses on these tasks.

Heterogeneous Data Analysis Task. Let a heterogeneous data-analysis task be represented as $x = (q, S)$, where q is a natural-language question, S is a finite set of allowed data sources that provides the context for data analysis, and y^* is the gold answer to question q grounded in the given data S. In real-world settings, the gold answer y^* is absent. A task is referred to as "heterogeneous" when the data source set S contains multiple modalities, such as databases, CSV files, JSON files, and document text.

Data Agent Harness. A data-agent harness H is a stateful program that aims to answer questions grounded in given data. It wraps a fixed language model m and determines what context the model sees and what actions the model takes at each step. For a data-analysis task x, a data-agent harness H generates a reasoning trajectory $\mathcal{T} = \{\tau_0, \tau_1, \ldots, \tau_i, y\} \sim \Pr_m(H, x)$, where y is the answer derived by the agent through the rollout trajectory $\{\tau_i\}$.

Optimization of Data Agent Harness. Suppose we have a development dataset $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$, where each instance D consists of a data-analysis task x and a gold answer y^*, i.e., $D = (x, y^*)$. Note that, in the wild, the gold answer y^* is absent. Given an evaluation mechanism $e(y, y^*)$, the objective of agent optimization is to find a data-agent harness design that maximizes the expected evaluation score:

$$H^* = \arg\max_H \mathbb{E}_{(x,y^*)\sim\mathcal{D},\mathcal{T}\sim\Pr_m(H,x)} e(y, y^*) \tag{1}$$

Agent Evaluation. To enable structured and verifiable scoring, both the agent answer and the ground truth are expressed in tabular form. Let $T_{\text{ans}} = \{C^1_{\text{ans}}, C^2_{\text{ans}}, \cdots\}$ and $T_{\text{gold}} = \{C^1_{\text{gold}}, C^2_{\text{gold}}, \cdots\}$ denote the agent answer and the ground truth, respectively, each comprising one or more columns. For each task pair (y, y^*), the evaluation score is computed as follows.

$$e(T_{\text{ans}}, T_{\text{gold}}) = \max(0, \underbrace{\frac{|T_{\text{ans}} \cap T_{\text{gold}}|}{|T_{\text{gold}}|}}_{\text{answer recall}} - \underbrace{\lambda \cdot \frac{|T_{\text{ans}} \setminus T_{\text{gold}}|}{|T_{\text{ans}}|}}_{\text{penalty}}), \tag{2}$$

Algorithm 1: Autonomous data-agent evolving

Input: Initial data-agent harness H_0 instantiated from the ReAct baseline, development task set $\mathcal{D}$, proposer P, optimization budget B
Output: Optimized data-agent harness $H^\star$

$\mathcal{A}^H \leftarrow \emptyset$
for $(x, y^*) \in \mathcal{D}$ **do**
  Collect artifacts (A^H_x), including predictions (y), execution logs, and reasoning traces, by running H_0 on task x
  Evaluate the answer $e_x = e(y, y^*)$
  $\mathcal{A}^H \leftarrow \mathcal{A}^H \cup \{(A^H_x, e_x)\}$
$S_{\text{best}} \leftarrow \text{mean}(\{e_x \mid x \in \mathcal{D}\})$
$H^\star \leftarrow H_0$
for $b = 1, \ldots, B$ **do**
  $H \leftarrow H^\star$
  Proposer P proposes reflections and improvement ideas by inspecting artifacts ($\mathcal{A}^H$)
  $H \leftarrow$ implement improvement ideas in H
  $\mathcal{A}^H \leftarrow \emptyset$
  for $(x, y^*) \in \mathcal{D}$ **do**
    Collect artifacts (A^H_x), including predictions (y), execution logs, and reasoning traces, by running H on task x
    Evaluate the answer $e_x = e(y, y^*)$
    $\mathcal{A}^H \leftarrow \mathcal{A}^H \cup \{(A^H_x, e_x)\}$
  $S_H \leftarrow \text{mean}(\{e_x \mid x \in \mathcal{D}\})$
  if $S_H \geq S_{\text{best}}$ **then**
    $S_{\text{best}} \leftarrow S_H$
    $H^\star \leftarrow H$
Return $H^\star$

where λ is set to 0.1 in this work. For the entire dataset, the final evaluation score is the average score across all tasks. This evaluation emphasizes that the primary objective is to recover every gold-value column appearing in the ground truth. Extra columns are undesirable but only lightly penalized.

3 AGENT EVOLVING

Autonomous data-agent evolving aims to upgrade the design of a data-agent harness through agent evaluation and LLM-based reflection, without human expert intervention, thereby improving performance on given data-analysis tasks. Because the development dataset contains diverse data-analysis tasks, agent evolving benefits from inter-task learning signals rather than over-tuning the harness to a single problem; that is, insights from solving one task can help solve others.

Autonomous data-agent evolving leverages Meta-Harness [6], a technique driven by a powerful agentic coding agent (e.g., Claude Code or Codex) with full access to data-agent artifacts such as reasoning traces, predicted answers, task-level evaluation scores, and error messages. The overall workflow is presented in Algorithm 1.

It takes as input an initial data-agent harness H_0 instantiated from the ReAct baseline [13], a development data-analysis task set $\mathcal{D}$, a proposer P, and an optimization budget B. Meta-Harness first executes H_0 on each task; records the prediction, execution log, reasoning trace, and evaluation score as task-level artifacts; and aggregates these artifacts to establish the initial best score. During each evolution round, the current best harness $H^\star$ is copied as a candidate harness H; the proposer inspects the accumulated artifacts to identify recurring failure patterns, generate reflections, and suggest harness-level improvements; and the suggested changes are implemented in the codebase. The revised harness is then rerun on the development tasks to collect a fresh artifact set and compute its aggregate score. If the revised harness matches or improves the best score, Meta-Harness accepts it as the new $H^\star$; otherwise, the previous best harness is retained while the new diagnostic artifacts continue to inform subsequent proposals. This closed loop couples evaluation, reflection, implementation, and rerunning, enabling autonomous cross-task improvement without manual expert intervention.

4 A STATIONARY POINT

Through autonomous agent-harness evolution, the system reaches a stationary point at which the data agent's performance stagnates. In this section, we examine the resulting harness design.

4.1 Agent Harness Overview

The final data-agent harness is organized around a lead agent, a set of specialist subagents, and multiple tools. As illustrated in Figure 1, once the lead agent receives an NL question, it first invokes the KnowledgeInjector subagent to check whether relevant background knowledge (e.g., database manuals or domain knowledge) is available. The subagent processes textual background material and injects only the parts most pertinent to the question into the agent memory. The lead agent then invokes the Planner subagent, which performs additional inspection of the given data sources and submits a high-level plan before execution. Guided by this plan, the lead agent, with full access to the toolkit, iteratively asks the model for tool calls, executes those calls, manages memory, and continues until an answer is accepted or the loop budget is exhausted. Once an answer is produced, the Verifier checks whether the current answer and execution trace appear semantically valid. If the answer or trace is corrupted, the Verifier diagnoses the failure and provides suggestions to the Planner and lead agent; otherwise, the AnswerNormalizer subagent projects the answer to a minimum safe column set to avoid the extra-column penalty in Eq. 2.

4.2 Sub-Agents

KnowledgeInjector. In addition to data sources that directly provide answers, there are often complementary documents that provide useful domain knowledge for data analysis or user guides for tools. For example, a database manual may describe the underlying database system used for storage and the SQL dialect that should be used for processing. However, for a given data-analysis task, only a small portion of such documents is usually useful. Injecting all background information into the LLM memory wastes valuable context space and, worse, can blur the model's focus. To address this issue, KnowledgeInjector extracts information highly relevant to the given question or summarizes the key points in the documents. The outputs of KnowledgeInjector are appended to the LLM memory.

Planner. The data agent is forced to think before acting. Planner aims to understand the data and task and to build a high-level action proposal for the agent. With such a plan, the agent operates under a guardrail and can act efficiently instead of repeatedly walking in circles. To produce a high-quality and implementable plan, Planner is equipped with multiple data-profiling tools for structured, semi-structured, and unstructured data. Specifically, for structured data such as databases and CSV files, these profiling tools provide not only basic metadata, such as column names and table sizes, but also exploratory data statistics, including inferred types, null percentages, distinct counts, common values, and numeric or date min/max values where applicable. For semi-structured data such as JSON files and unstructured data such as documents, the profiling tools return record previews or text slices, helping the agent understand schemas and determine appropriate data-processing approaches.

Verifier. Although Planner provides guidance for data processing, it does not always guarantee that the agent strictly follows the plan at every step. Verifier provides an additional safety check when the data agent believes it has produced a trustworthy answer to the question. Verifier receives (1) the original NL question, (2) the plan produced by Planner, (3) the action trace including tool calls and tool results, and (4) the produced answer. Internally, Verifier compresses long traces into several checkpoints to reduce LLM context consumption. Because ground truth is unavailable at inference time, Verifier performs a soft evaluation of the candidate answer by rating it according to the following rubrics:

- **Plan validity**. Does the plan satisfy all requirements in the question?
- **Filter coverage**. Does the trace apply every constraint in the question as planned?
- **Cross-source coverage**. Does the agent join all necessary sources on demand as planned?
- **Aggregation correctness**. Does it use the correct count, average, sum, percentage, or formula?
- **Answer-format sanity**. Are there truncated strings, fake placeholders, suspiciously empty results, or implausible row counts?

If the verdict is accept, Verifier returns the answer; if the verdict is reject, Verifier returns hints and feedback to the lead agent, which then spends additional effort addressing potential errors.

AnswerNormalizer. Motivated by the penalty on redundant information in answers, AnswerNormalizer converts a possibly overwide candidate table into the minimum table shape expected by the benchmark, without fabricating values or changing row cardinality. AnswerNormalizer builds a mapping from question intent to output schema: (1) if the question asks for a single quantity (e.g., "how many", "what is the percentage", "what is the average", "state the X", or "what was Y"), the output contains exactly one column; (2) if the question lists multiple fields (e.g., "name and website" or "score and date"), the output includes exactly those fields, and helper columns are dropped. AnswerNormalizer invokes an LLM to decide which pattern the output should follow. Because missing a desired column is penalized more heavily than keeping an extra column,

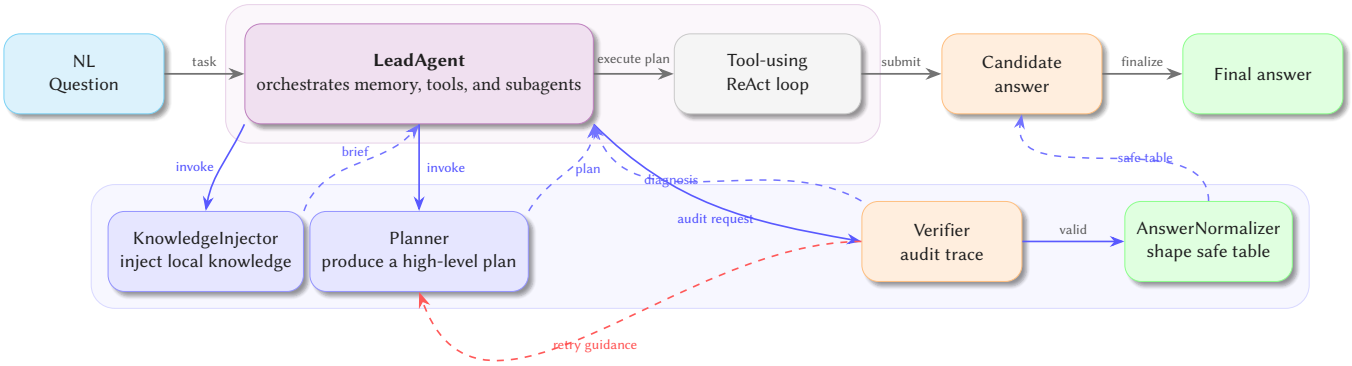

Figure 1: Lead-agent coordination workflow. The lead agent receives the question, invokes specialist subagents for knowledge injection and planning, executes the plan through a bounded ReAct tool loop, and then invokes verification and answer normalization before writing the final prediction. Dashed arrows indicate feedback used for diagnosis and retry.

AnswerNormalizer remains conservative and does not blindly trust the LLM projection. It refuses risky column drops in several cases:

- If the question clearly asks for multiple fields, it does not drop non-internal columns.
- If the LLM's confidence is low and any column would be dropped, it keeps the original answer.
- If a dropped column has mostly distinct values and confidence is not high, it keeps the original answer.
- If a dropped column looks like a real deliverable value, such as a URL, ISO date, person/place name, or non-flag numeric metric, it keeps the original answer.

4.3 Tools & Memory Management

The data-agent harness is equipped with tools commonly used in general code-agent products such as Codex and Claude Code, including file read/write, a Python interpreter, grep, glob, and related utilities. In addition, it is equipped with a SQL executor and the data-profiling tools introduced above. Together, these tools provide broad capabilities for exploring, understanding, and processing data.

For agent memory, the harness adopts an append-only strategy that adds new messages to the tail of the context to facilitate KV caching. To avoid excessive and stale agent memory, it adopts a three-tier autocompaction mechanism similar to that used in Claude Code.

(1) **Drop low-value failures**: Remove old error messages and all-failed tool-call/result pairs from the middle of the trace.
(2) **Truncate old large tool results**: Shorten long old tool outputs. This preserves schemas, examples, and final context while reducing token load.
(3) **Summarize old turns**: If the conversation is still too large, summarize older turns with the LLM. The summary is factual and includes the original task, key data observations, decisions made, and pending subgoals.

Whenever new context is concatenated, the agent estimates the token consumption of its memory. If the consumption is about to exceed the LLM's context window, the agent triggers memory autocompaction tier by tier.

5 EXPERIMENTS

5.1 Setup

Dataset. We use the open dataset released by the 2026 KDD Cup[1] as the test set. It contains 50 data-analysis tasks, each consisting of an NL question, multiple heterogeneous data files, and a gold answer for evaluation.

Compared Systems. We compare different versions of the data agent using a fixed LLM, Qwen3.5-35b-a3b. We choose this relatively small model because a smaller LLM benefits more from a strong agent harness. In addition, we evaluate the best evolved harness with a frontier model, Qwen3.7-max. We also evaluate Codex, a SOTA coding-agent product, on the same data-analysis tasks.

5.2 Comparison across Iterations

5.2.1 Data-agent harness self-evolving. Figure 2 reports the raw evaluation score obtained after each meta-harness iteration. The initial ReAct-style baseline achieves only 0.359, showing that a generic tool-using loop is insufficient for heterogeneous data-analysis tasks. The largest single jump appears in the first iteration, where adding an explicit Planner before execution raises the score from 0.359 to 0.527, a 0.168 absolute improvement. This indicates that many early failures come from acting before understanding the data landscape: once the agent is forced to inspect available sources, reason about likely joins or transformations, and commit to a high-level plan, it avoids a large fraction of blind or premature tool calls.

The next group of improvements further stabilizes the execution loop. Fixing planner inspection and plan-generation errors increases the score to 0.566. Adding memory autocompaction further raises it to 0.606, suggesting that long traces and context pressure are practical bottlenecks even when the reasoning policy is improved. Canonicalizing errors and standardizing tool-return values

[1] https://dataagent.top/

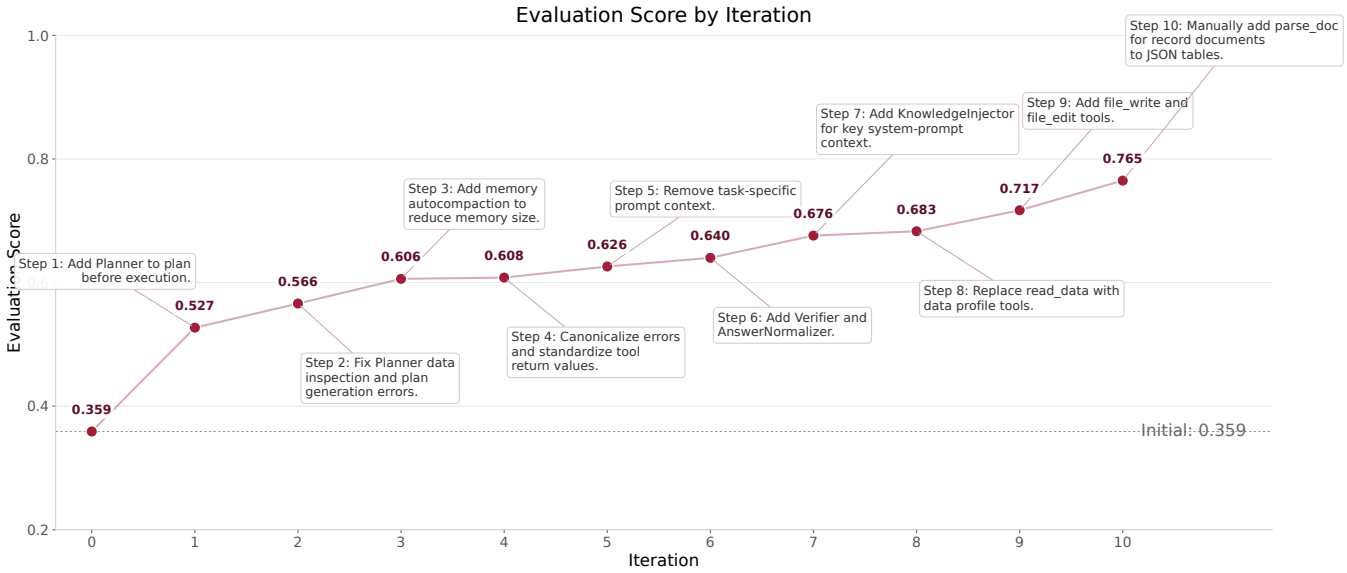

Figure 2: Performance evolution of the data-agent harness across iterative meta-harness improvements.

yields a nearly flat but non-negative change from 0.606 to 0.608, while removing task-specific prompt context increases the score to 0.626. These changes do not add new analytical capability by themselves; instead, they reduce harness friction. Cleaner tool outputs make observations easier for the model to consume, memory compaction keeps long traces within the usable context window, and removing task-specific prompt assumptions improves generality across the 50 tasks.

The middle iterations show the effect of adding specialist finalization and grounding components. Introducing Verifier and AnswerNormalizer improves the evaluation score from 0.626 to 0.640 by catching semantically suspicious answers and reducing unsafe answer-table shapes before writing the final result. Adding KnowledgeInjector then raises the score to 0.676, showing that task-local manuals, domain rules, or hidden definitions are important for a non-trivial subset of tasks. Replacing generic data reading with data-profile tools yields a smaller gain to 0.683, because the agent receives compact statistics, schemas, and previews rather than only raw snippets, making source selection and transformation planning more reliable.

The final iterations improve the long tail of heterogeneous tasks. Adding file-writing and editing tools increases the score from 0.683 to 0.717, indicating that some tasks benefit from materializing intermediate artifacts rather than keeping all derived data inside transient model responses.

5.2.2 Further manual engineering. The only improvement outside the autonomous meta-harness loop is the manually proposed `parse_doc` tool, which increases the score from 0.717 to 0.765. This tool is designed for record-structured Markdown or text documents. In the benchmark, we observed that some documents contain many repeated entity blocks, such as one patient, case, product, or event per paragraph, and each block is usually anchored by a stable identifier. Such regularity is difficult for the LLM to exploit from a short document preview alone: the agent may notice a few examples but fail to infer that the entire document can be treated as a latent table.

The `parse_doc` tool converts this hidden semi-structured pattern into an explicit JSON table. It first reads the source document and splits it into paragraph-like record chunks. It then asks the model to infer the entity-ID pattern, groups all text belonging to the same ID, infers an extraction schema from the task question and target information need, and extracts one structured JSON record for each entity. The extracted records are type-coerced and saved under the agent's workspace, after which the normal data tools can process them through `read_json`, `execute_python`, and so on. In effect, `parse_doc` turns a long document into an auditable intermediate table that can be filtered, joined, and aggregated like other structured data, which faciliates data manipulation via coding.

This manual addition targets a failure mode not fully addressed by the evolved harness: document previews help the agent orient itself, but they do not guarantee complete extraction from long, repetitive documents. By adding a tool that explicitly recognizes repeated record structure, the harness can collect key information that would otherwise remain buried in prose. The final gain of 0.048 suggests that document-to-table conversion is important for the long tail of mixed-source data-analysis tasks, especially when the correct answer depends on combining textual records with structured files.

Overall, the final harness improves by 0.406 over the 0.359 baseline and achieves a 2.13× relative score increase. The curve supports the central claim of the paper: autonomous data-agent performance improves most when the meta-harness translates recurring failure buckets into concrete harness mechanisms—planning, grounding, memory management, verification, answer shaping, profiling, and document parsing—rather than relying only on broader prompts or a larger model. Such observations indicate the promising potential of data agent self-evolution.

Table 1: Comparison with SOTA systems.

System	Evaluation score
Final data agent + Qwen3.5-35b-a3b	0.765
Final data agent + Qwen3.7-max	0.756
Codex + GPT-5.5	0.676

5.3 Comparison with SOTA systems

Table 1 compares the final evolved data-agent harness with two stronger system-level baselines. The best score is achieved by our final harness with Qwen3.5-35b-a3b, reaching 0.765. Replacing the backend model with the frontier Qwen3.7-max gives a very close score of 0.756, while Codex with GPT-5.5 obtains 0.676. Thus, the evolved harness with the smaller fixed model outperforms Codex by 8.9 absolute points and remains slightly ahead of the same harness using the stronger Qwen3.7-max backend.

The first finding is that harness design can dominate raw model strength in this benchmark. Qwen3.5-35b-a3b is not expected to be stronger than a frontier coding-agent setup in general-purpose reasoning or coding, yet the final harness obtains the highest score. This suggests that heterogeneous data analysis is not merely a model-capability problem. The task requires reliable source discovery, bounded planning, controlled tool execution, memory management, document-to-table extraction, semantic verification, and final answer shaping. These operations are externalized as harness mechanisms in our system, so the model receives a more structured problem at each stage instead of being asked to solve the entire benchmark through an unconstrained monolithic loop.

The second finding is that a stronger model does not automatically improve the final result when the harness already constrains the task effectively. The Qwen3.7-max variant scores 0.756, only 0.009 below the Qwen3.5 variant. This small gap indicates that the final harness is relatively model-robust: once the agent is grounded with knowledge injection, data profiling, planning, verification, and answer normalization, the remaining errors are less about general language-model intelligence and more about benchmark-specific edge cases, tool-use trajectories, and exact final-table construction. It also suggests that a larger model may occasionally make different high-level choices or follow verifier feedback differently, causing small regressions even when its base capability is stronger.

The third finding concerns the comparison with Codex. Codex is a strong general coding agent, but its default workflow is optimized for software-engineering tasks rather than scored heterogeneous data analysis. In this benchmark, final answers are tabular artifacts, and small mistakes in row cardinality, column selection, value formatting, source coverage, or document extraction can sharply reduce the score. Our data-agent harness explicitly targets these failure modes through data-profile tools, conservative answer normalization, verifier checks, and the manually added `parse_doc` tool. The 0.765 versus 0.676 gap therefore supports the central design motivation: data-agent systems require domain-specific runtime structure, not only a capable code-generation agent.

Overall, these results show that the evolved harness transfers the burden of reliability from the model alone to a coordinated system of specialist subagents and deterministic guards. The comparison also explains why meta-harness evolution is valuable: by repeatedly converting observed benchmark failures into concrete harness components, the final system can outperform a general-purpose SOTA coding agent and remain competitive across backend models.

6 DISCUSSION

Although the data-agent harness is effective, we observe several obstacles that prevent self-evolution from improving further.

Overfitting. As the data agent improves itself by learning from failure cases, it can easily overfit to those cases. For example, in our experiments, the agent may add case-specific hints to the system prompts, forcing the model to attend to context that was previously ignored. Such modifications are generally not useful, and worse, they can degrade the data agent.

Local optimum. In our experiments, data-agent harness self-evolution may stagnate in later evolution steps (after 9 iterations in our experiment). It often becomes stuck in cycles of adding harness modifications and then reverting or redoing them. Such stagnation does not mean that the harness has reached the best possible design: as observed, simply adding a document-parsing tool significantly improves data-agent performance.

7 RELATED WORK

Agentic data analysis. Autonomous data analysis has become a promising research direction and has been studied from multiple perspectives. (1) NL2SQL [5, 9] and semantic data analysis [12, 16]. These works bridge LLMs and database systems, enriching the functionality that databases can support and lowering the barrier for non-technical users. (2) LLM-wise autonomous data analytics [4, 8, 15]. These works aim to enhance base-model capabilities for data processing by fine-tuning models on real or synthetic training data with specific training regimes. (3) Agent-wise autonomous data analytics [3, 7]. Another line of work focuses on agent-harness design to ensure that an LLM can execute data-analysis tasks correctly and precisely.

LLM system self-evolution. Motivated by the idea that intelligent systems can teach and improve themselves, a line of research studies self-optimization for LLM systems. For example, TextGrad [14] and GEPA [2] achieve self-improvement of textual ingredients, such as prompts, through reflective prompt optimization. Meta-Harness [6] and Optimize Anything [1] extend similar ideas to broader artifacts, such as agent harnesses and code, that can be expressed as text.

8 CONCLUSION

This paper studied autonomous data-agent evolution through a Meta-Harness workflow that turns benchmark feedback, execution traces, and recurring failure modes into concrete harness improvements, eventually yielding a multi-agent data-agent harness. Our experiments show that, through self-evolution, the data agent achieves consistent performance improvements from 0.356 to 0.717. These results suggest a practical direction for future agentic data systems: treating failures as reusable design signals and combining flexible LLM reasoning with deterministic guards, specialist tools, and auditable intermediate artifacts.

REFERENCES

[1] Lakshya A Agrawal, Donghyun Lee, Shangyin Tan, Wenjie Ma, Karim Elmaaroufi, Rohit Sandadi, Sanjit A. Seshia, Koushik Sen, Dan Klein, Ion Stoica, Joseph E. Gonzalez, Omar Khattab, Alexandros G. Dimakis, and Matei Zaharia. 2026. optimize_anything: Unified Text Optimization can Outperform Specialized Systems. *CoRR* abs/2605.19633. https://doi.org/10.48550/arXiv.2605.19633

[2] Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Daniel Klein, Matei Zaharia, and Omar Khattab. 2025. GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning. *CoRR* abs/2507.19457 (2025). https://doi.org/10.48550/arXiv.2507.19457

[3] Meihao Fan, Ju Fan, Nan Tang, Lei Cao, Guoliang Li, and Xiaoyong Du. 2025. AutoPrep: Natural Language Question-Aware Data Preparation with a Multi-Agent Framework. *Proc. VLDB Endow.* 18, 10 (2025), 3504–3517.

[4] Meihao Fan, Ju Fan, Yuxin Zhang, Shaolei Zhang, Xiaoyong Du, Jie Song, Peng Li, Fuxin Jiang, Tieying Zhang, and Jianjun Chen. 2026. DeepPrep: An LLM-Powered Agentic System for Autonomous Data Preparation. *CoRR* abs/2602.07371 (2026). https://doi.org/10.48550/arXiv.2602.07371

[5] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proc. VLDB Endow.* 17, 5 (2024), 1132–1145. https://www.vldb.org/pvldb/vol17/p1132-gao.pdf

[6] Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. 2026. Meta-Harness: End-to-End Optimization of Model Harnesses. *CoRR* abs/2603.28052 (2026). https://doi.org/10.48550/arXiv.2603.28052

[7] Boyan Li, Yiran Peng, Yupeng Xie, Sirong Lu, Yizhang Zhu, Xing Mu, Xinyu Liu, and Yuyu Luo. 2026. DeepEye: A Steerable Self-driving Data Agent System. *CoRR* abs/2603.28889 (2026). https://doi.org/10.48550/arXiv.2603.28889

[8] Fengyu Li, Junhao Zhu, Kaishi Song, Lu Chen, Zhongming Yao, Tianyi Li, and Christian S. Jensen. 2026. Replacing Multi-Step Assembly of Data Preparation Pipelines with One-Step LLM Pipeline Generation for Table QA. *CoRR* abs/2602.22721 (2026). https://doi.org/10.48550/arXiv.2602.22721

[9] Yin Lin, Tianjing Zeng, Zhongjun Ding, Rong Zhu, Bolin Ding, H. V. Jagadish, and Jingren Zhou. 2026. SEMA-SQL: Beyond Traditional Relational Querying with Large Language Models. *CoRR* abs/2604.23477 (2026). https://doi.org/10.48550/arXiv.2604.23477

[10] Shu Liu, Shubham Agarwal, Monishwaran Maheswaran, Mert Cemri, Zhifei Li, Qiuyang Mang, Ashwin Naren, Ethan Boneh, Audrey Cheng, Melissa Z. Pan, Alexander Du, Kurt Keutzer, Alexandros G. Dimakis, Koushik Sen, Matei Zaharia, and Ion Stoica. 2026. EvoX: Meta-Evolution for Automated Discovery. *CoRR* abs/2602.23413 (2026). https://doi.org/10.48550/arXiv.2602.23413

[11] Ruiying Ma, Shreya Shankar, Ruiqi Chen, Yiming Lin, Sepanta Zeighami, Rajoshi Ghosh, Abhinav Gupta, Anushrut Gupta, Tanmai Gopal, and Aditya G. Parameswaran. 2026. Can AI Agents Answer Your Data Questions? A Benchmark for Data Agents. *CoRR* abs/2603.20576 (2026). https://doi.org/10.48550/arXiv.2603.20576

[12] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G. Parameswaran, and Eugene Wu. 2025. DocETL: Agentic Query Rewriting and Evaluation for Complex Document Processing. *Proc. VLDB Endow.* 18, 9 (2025), 3035–3048. https://www.vldb.org/pvldb/vol18/p3035-shankar.pdf

[13] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *ICLR*. OpenReview.net. https://openreview.net/forum?id=WE_vluYUL-X

[14] Mert Yüksekgönül, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. 2025. Optimizing generative AI by backpropagating language model feedback. *Nat.* 639, 8055 (2025), 609–616.

[15] Shaolei Zhang, Ju Fan, Meihao Fan, Guoliang Li, and Xiaoyong Du. 2025. DeepAnalyze: Agentic Large Language Models for Autonomous Data Science. *CoRR* abs/2510.16872 (2025). https://doi.org/10.48550/arXiv.2510.16872

[16] Junhao Zhu, Lu Chen, Xiangyu Ke, Ziquan Fang, Tianyi Li, Yunjun Gao, and Christian S. Jensen. 2025. Beyond Relational: Semantic-Aware Multi-Modal Analytics with LLM-Native Query Optimization. *CoRR* abs/2511.19830 (2025). https://doi.org/10.48550/arXiv.2511.19830