TexeraAgent: An AI-Agent for Data Science Using Dataflows

Jiadong Bai
University of California, Irvine
Irvine, California, USA
jiadongb@ics.uci.edu

Yicong Huang
University of Massachusetts Amherst
Amherst, Massachusetts, USA
yiconghuang@cs.umass.edu

Chen Li
University of California, Irvine
Irvine, California, USA
chenli@ics.uci.edu

Abstract

Recent advances in large language models (LLMs) have made human-agent collaboration a promising and powerful paradigm for data science. Among existing LLM-based solutions, conversational analytics is convenient but makes it difficult for users to inspect intermediate steps and verify errors. Script-based ReAct improves transparency by grounding interaction in executable scripts, but still inherits the complexity and low-level details of programming. In this paper, we present a novel ReAct-based AI agent called TexeraAgent in the Apache Texera system. This agent is built on the idea of using dataflow as the abstraction for human-agent collaboration. Such an abstraction is well-suited to LLMs' internal reasoning on data science tasks. We will show the benefits of TexeraAgent, including how an analyst easily understands the agent's behaviors through an intuitive dataflow-based interface, how the analyst efficiently inspects the agent's past actions to provide feedback, and how the agent can finish a task with high accuracy and low cost.

Keywords

Human-AI collaboration, Data Science, Agent, LLM, Dataflow

1 Introduction

Recent advances in large language models (LLMs) have made human-agent collaboration increasingly promising for data science. A common approach is conversational analytics, where human analysts use natural language to ask agents to explore data, generate plots, and summarize findings, using LLM services such as ChatGPT and Claude. While convenient, its interface typically limits human analysts to high-level supervision [4] and provides little visibility into the underlying data or intermediate steps. As a result, hallucinations and analysis errors are difficult to avoid and verify [5], and the human-agent interaction becomes hard to manage for complex, iterative, and multi-step tasks. Recent systems have shifted toward a "Script ReAct" paradigm, e.g., Databricks Genie Code [3], in which a human analyst and the agent collaborate on executable code and notebooks. As shown in the top part of Figure 1, an analyst in such a system describes the task in natural language, and the agent generates executable scripts (often in the form of notebook cells) that serve as the shared artifact for collaboration. By grounding interaction in executable artifacts, Script ReAct offers greater transparency and control than conversational analytics [12].

While more powerful, Script ReAct still inherits the following limitations of a programming interface. (1) *Human barrier and cognitive overhead.* Programming languages are not an ideal medium for analysts: they raise the entry barrier for non-programmers, and even experienced programmers must spend effort reading code before they can verify or revise the intended analysis. (2) *Weak artifact traceability.* While code itself can be versioned, its outputs often live in ephemeral artifacts (in-memory states, temporary files, visualizations, console logs), making it hard to track dependencies and maintain provenance across iterations. (3) *Noisy agent context.* Low-level code details (syntax, variables, control flow) inflate token cost, degrade grounding over long interactions, and increase the risk of hallucinated steps.

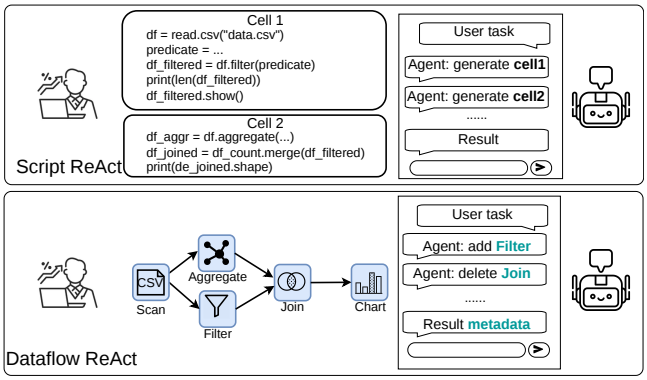

Figure 1: Script ReAct versus Dataflow ReAct (TexeraAgent, this work) on data science tasks.

In this paper, we present a novel idea to leverage LLMs in data science based on the following insight. Before the emergence of LLM agents, the data management and workflow communities already developed dataflow systems as an effective *abstraction* for complex data analytics. In these systems, computation is represented as a graph of operators connected by data dependencies (often as a directed acyclic graph or DAG) [1, 9]. This abstraction is appealing because it naturally exposes the semantic structure of an analysis while suppressing low-level implementation details [1, 18]. Even relational DBMS systems expose execution plans through interfaces such as `EXPLAIN` in PostgreSQL. Such an abstraction is well-suited to LLMs' internal reasoning on data tasks.

Based on this insight, our idea is to use dataflow as the primary abstraction for human-agent collaboration in data science tasks. We call this approach *Dataflow ReAct*, as shown in the bottom part of Figure 1. Compared to Script ReAct, we replace code with dataflow as the collaboration medium, so that both the analyst and the agent reason over a higher-level analytical structure rather than low-level programming details. We present TexeraAgent, an LLM-based agent on top of the Apache Texera[1] system. The agent can collaborate with analysts through interactive dataflows rather than through script-based notebooks. Specifically, TexeraAgent consists of an interactive dataflow UI, a chatbox, a stateless dataflow-execution

SAO Workshop at CAIS'26, May 26, 2026, San Jose, California, USA

[1] Currently under Apache incubation.

engine, and a catalog-governed lakehouse. The agent understands an analyst's requests, constructs corresponding dataflows, presents them to the analyst for inspection, and submits them to the execution engine. It captures dataflow-aware contexts and keeps track of context changes through governance versions.

In the paper, we show how TexeraAgent enables human analysts to better trace its analytical process, inspect the rationale behind intermediate decisions, and intervene with effective feedback during task execution. Our preliminary benchmark further highlights that TexeraAgent achieves a score of 79% for evaluated data science tasks, whereas a script-based ReAct baseline achieves a score of 68%.

2 System Overview

Apache Texera. It is an open-source system for human-AI collaborative data science using visual workflows. It enables analysts to construct, execute, and refine data analysis tasks through an intuitive GUI, assisted by AI agents that understand natural-language instructions. Texera is well suited for a wide range of applications, including "AI for Science," making advanced AI and data science capabilities accessible to a broader community. It can run on a laptop for local use or be deployed in the cloud to support scalable processing of large datasets. Started in 2016 and now under Apache incubation, the system supports parallel execution of data science and AI/ML workflows, real-time collaboration, Python/Java UDFs, and decoupled compute-storage for cloud deployment. More details are available at the project site [2].

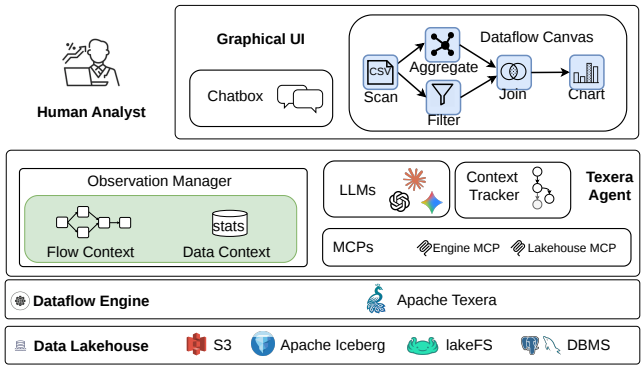

Figure 2: TexeraAgent system architecture.

Figure 2 illustrates the architecture of the system with TexeraAgent. It consists of four layers: a graphical user interface (GUI), the LLM-based *TexeraAgent*, a dataflow engine, and a data lakehouse. At the top, the GUI includes a chatbox for analysts to communicate with *TexeraAgent* in natural language, and a canvas for visualizing a dataflow. The canvas exposes the high-level structure of the dataflow, while optionally allowing human analysts to inspect operator-level details such as code and property values (e.g., a threshold of a filter operator). At the bottom, the dataflow engine executes a dataflow, while the data lakehouse stores data such as CSV files, generated artifacts, execution output, and runtime logs.

At the core of the system is the *TexeraAgent*, which coordinates reasoning, execution, and observation. It connects to one or more

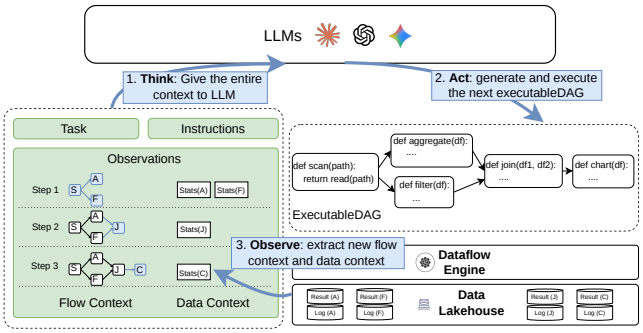

Figure 3: The "Think, Act, and Observe" loop for Dataflow ReAct used by TexeraAgent.

LLMs, and interacts with the dataflow engine and data lakehouse through Model Context Protocols (MCPs). In particular, the engine MCP allows the agent to edit and inspect the dataflow and submit the dataflow for execution, while the lakehouse MCP allows the agent to retrieve metadata and artifacts from the storage. Within the agent, an *Observation Manager* maintains two dataflow-oriented context abstractions, namely "Flow Context" and "Data Context." A "Context Tracker" versions these contexts together with other generated artifacts across iterations.

Human-Agent Collaboration with Dataflow ReAct Loop. When the analyst provides a task through the chatbox, the agent starts the Dataflow ReAct loop, as illustrated in Figure 3. The agent takes multiple steps to complete the task, and each step contains *Think*, *Act*, and *Observe*. During thinking, the agent packages the task, the latest observations, and instructions for using MCPs into a context, and sends it to an LLM to reason about the next move. The LLM generates changes to the previous dataflow. The agent acts to update the dataflow shown in the UI and submits the new *ExecutableDAG* to the engine for execution. Once the execution completes, the agent collects observations on both the new dataflow and the execution output and saves them in the data lakehouse. The agent repeats this Think-Act-Observe loop until it has made sufficient progress on the task. Throughout the process, the analyst remains in the loop and can monitor agent-driven changes to the dataflow, inspect execution results and runtime information, interrupt execution, revise the dataflow directly, or provide feedback to the agent to redirect subsequent reasoning and actions.

Observations Designed for Dataflow ReAct. As illustrated in Figure 3, the *Observe* step is designed not to feed raw code or low-level runtime states to *TexeraAgent*, but to extract two high-level, dataflow-oriented observations, namely *Flow Context* and *Data Context*. The Flow Context summarizes the structure and semantics of the analysis as a dataflow, including operators, dependencies, connectivity, lineage, and stage-level semantic roles. In Figure 3, after step 2, the agent extracts the newly added operator *Chart* together with its incoming edge from *Join* as part of the update to Flow Context in step 3. The Data Context summarizes properties of the data and generated artifacts, including schema, metadata, basic statistics, distributions, and profiles of intermediate and final results. In Figure 3, after step 2, the agent fetches the schema and

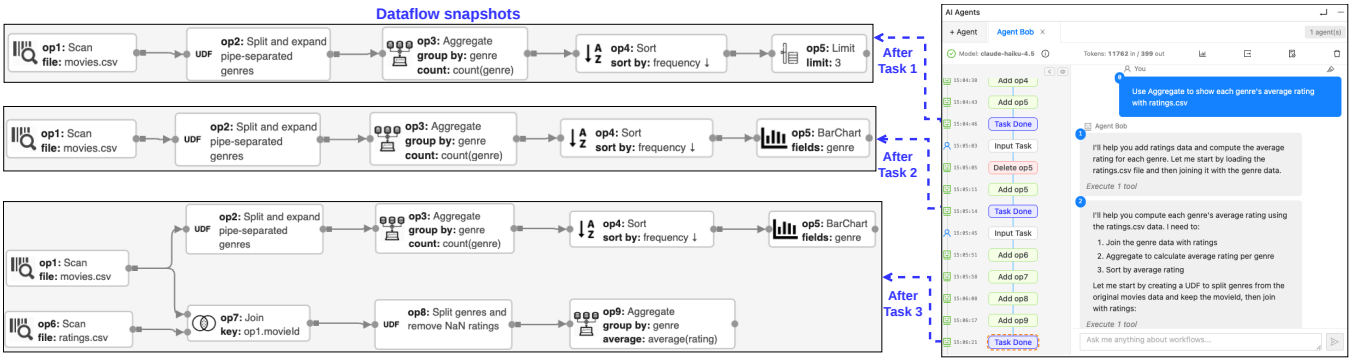

Figure 4: The TexeraAgent interface across three tasks. The dataflow canvas (left) shows the dataflow evolving in real time with operator-level semantics. The chatbox (right) displays the agent's reasoning and keeps track of context versions.

statistics of the newly produced *Result(C)* as part of the update to the Data Context in step 3.

Context Tracking with Governance Versions. To support iterative collaboration, the Context Tracker versions artifacts and contexts throughout the interaction. Every analyst- or agent-driven dataflow modification is captured as an event. Each action-carrying event produces a snapshot: a dataflow DAG with its Flow Context, Data Context, and references to per-operator results. Events form an event chain and snapshots form a snapshot tree, both kept in memory as the structural backbone of a session. The referenced results are persisted in the lakehouse as Apache Iceberg tables over columnar (e.g., Parquet) files, and reused across snapshots: when a new snapshot executes, the engine skips operators whose result references carry over and computes only those introduced or modified by the latest action. The analyst can view the event chain on the GUI and time-travel between events: selecting one retrieves its snapshot and restores the dataflow and its results on the canvas. In this way, TexeraAgent manages ReAct sessions with Git-like branching and time-travel for human-agent collaboration on data science, complementing source-code versioning and Git-for-data style data-lake versioning [16, 17].

3 Example Scenarios

We use a data science example to explain TexeraAgent, where an analyst named Alice wants to study how genres relate to movie ratings stored in two files, movies.csv and ratings.csv, which share a movie-ID column.

3.1 Analyst Monitoring and Understanding Agent's Behavior through a Dataflow

Initially, Alice wants to gain insights about the genres of the movies and sends the following task to the agent:

Task 1: *"Compute the top three genres of movies.csv."*

While *TexeraAgent* iterates through the Dataflow ReAct loop, Alice can follow the agent's progress in real time through the canvas, which renders the latest dataflow after each iteration. Figure 4 shows the dataflow snapshot after this task is finished: a scan on `movies.csv`, a Python UDF that splits nested genres into separate records, then aggregate, sort, and limit operators to count, rank, and keep the top three genres.

Next, Alice wants to visualize the genre distribution and sends the following task:

Task 2: *"Show the genre distribution instead."*

The agent resumes from task 1's context, with the version timeline showing continuity between the two tasks. It takes two ReAct steps: it removes the limit operator (recognizing Alice now wants the full distribution rather than the top three) and adds a bar-chart operator to display the genre distribution. Next, Alice sends a new task to relate ratings to genres:

Task 3: *"Calculate each genre's average rating with ratings.csv."*

The rest of Figure 4 shows the dataflow and chatbox after task 3. The agent adds a scan on `ratings.csv` and joins it with the movies table on the movie-ID column. Noticing many "NaN" values in the join's statistics, it connects a Python UDF to drop those rows before aggregating the average rating per genre.

3.2 User Inspecting Agent's Past Actions to Provide Feedback

Alice notices many "NaN" values in the final output and wants to confirm whether they come from the join or the original ratings file. From the join's rating-column statistics, she sees about 20% of records contain "NaN," and traces the dataflow downstream to find the UDF that drops them. She prefers to replace those values with zero so genres' ratings are not inflated, and clicks the past action that introduced the UDF:

Feedback: *"Treat all the NaN ratings as 0 instead."*

The agent recognizes this as a response to a past action and uses the Context Tracker to branch from the version after the UDF was introduced, reassembling its detached context (dataflow, results, observations) and combining it with Alice's feedback to start a new ReAct loop. The LLM returns an action that modifies the UDF to replace "NaN" with 0, as shown in Figure 5. *TexeraAgent* continues on this new branch while keeping the original available for inspection. After a few more steps, Alice sees the genres' ratings drop by 0.6.

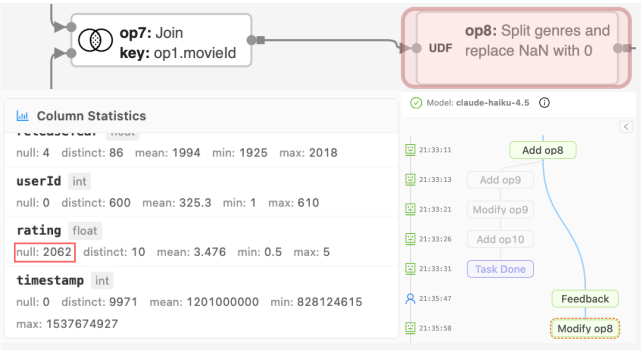

Figure 5: Alice inspects the statistics of the join operator's result, identifies several NaN ratings, and provides feedback to the UDF operator. *TexeraAgent* branches out to revise the operator.

3.3 Higher Accuracy and Lower Cost

The dataflow abstraction also benefits the agent itself: the Flow Context and Data Context summarize each step's effect at the operator and schema level, which compresses observations and removes the syntax-, variable-, and stack-trace noise that otherwise crowds the agent's context window across iterations.

To evaluate this benefit, we ran the 104 tasks of KramaBench [13], an end-to-end data-science benchmark over real data lakes spanning six domains (archeology, astronomy, biomedical, environment, legal, wildfire) with 1,764 files. Each task provides a natural-language question and the associated raw data files, and the system must produce a final typed answer. We compared TexeraAgent with a Script ReAct baseline built on smolagents [15]; both agents ran under KramaBench's Oracle input mode with the same LLM (GPT-5-mini or GPT-5.2) and the same iteration limit, the only difference being scripts versus dataflows. We follow KramaBench's end-to-end automation protocol, which scores each task in $[0, 1]$ using answer-type-specific metrics and reports the mean across all tasks. As shown in Table 1, TexeraAgent achieves a higher mean score at lower cost on both LLMs; on GPT-5-mini, Script ReAct reaches 63.5% at \$0.0204/task, while TexeraAgent reaches 76.9% at \$0.0168/task. Due to the page limit, we will discuss the detailed setup and the analysis of why Dataflow ReAct outperforms Script ReAct in a follow-up paper.

Table 1: Comparison results on KramaBench [13]

Agent	LLM			
	GPT-5-mini		GPT-5.2	
	Score (%)	Cost/task ($)	Score (%)	Cost/task ($)
Script Agent	63.5	0.0204	68.3	0.0964
TexeraAgent	76.9	0.0168	79.1	0.0843
Improvements	+13.4	-17.6%	+10.8	-12.6%

4 Related Work

Script-based data science agents such as DA-Agent [8] and DS-Agent [6] generate and execute Python scripts in a ReAct loop. Because code serves as both the execution artifact and the agent context, these agents suffer from noisy contexts filled with low-level code details such as syntax, variables, and their values after execution. Multi-agent frameworks such as Data Interpreter [7] and DS-STAR [14] decompose tasks across specialized sub-agents for better planning and error recovery, but their reasoning remains script-driven, inheriting the same context noise with additional multi-agent orchestration overhead. Both single-agent and multi-agent approaches also provide limited or no interfaces for analysts, preventing effective human-agent collaboration on tasks requiring human feedback. LLM-powered systems such as SiriusBI [10] and DataChat [11] do offer interactive interfaces, but they internally still generate scripts as context and expose them through chat-based or notebook-like interfaces. Our work differs from these studies by using dataflow as the primary abstraction in human-agent collaboration, offering the aforementioned benefits.

Acknowledgments

This work was supported by NIH NIDDK award 2U24DK097771-11 and a grant from SAP.

References

[1] Tyler Akidau, Robert Bradshaw, Craig Chambers, et al. 2015. The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing. *VLDB* (2015).
[2] Apache Texera. 2026. Apache Texera. https://texera.apache.org/.
[3] Databricks. 2026. Use Genie Code for data science. https://docs.databricks.com/aws/en/notebooks/ds-agent. Accessed: 2026-03-27.
[4] Ian Drosos et al. 2024. "It's like a rubber duck that talks back": Understanding Generative AI-Assisted Data Analysis Workflows through a Participatory Prompting Study *(CHIWORK '24)*.
[5] Ken Gu, Ruoxi Shang, Tim Althoff, et al. 2024. How Do Analysts Understand and Verify AI-Assisted Data Analyses? *(CHI '24)*.
[6] Siyuan Guo, Cheng Deng, Ying Wen, et al. 2024. DS-Agent: Automated Data Science by Empowering Large Language Models with Case-Based Reasoning *(Proceedings of Machine Learning Research)*.
[7] Sirui Hong, Yizhang Lin, et al. 2025. Data Interpreter: An LLM Agent for Data Science. In *ACL*.
[8] Yiming Huang, Jianwen Luo, Yan Yu, et al. 2024. DA-Code: Agent Data Science Code Generation Benchmark for Large Language Models.
[9] Michael Isard, Mihai Budiu, Yuan Yu, et al. 2007. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks.
[10] Jie Jiang, Haining Xie, et al. 2025. SiriusBI: A Comprehensive LLM-Powered Solution for Data Analytics in Business Intelligence. *Proc. VLDB Endow.* (2025).
[11] Rogers Jeffrey Leo John et al. 2023. DataChat: An Intuitive and Collaborative Data Analytics Platform *(SIGMOD '23)*. 203–215.
[12] Majeed Kazemitabaar et al. 2024. Improving Steering and Verification in AI-Assisted Data Analysis with Interactive Task Decomposition *(UIST '24)*.
[13] Eugenie Lai et al. 2025. KramaBench: Evaluating End-to-End Data-Science Agents.
[14] Jaehyun Nam et al. 2026. DS-STAR: Data Science Agent for Solving Diverse Tasks across Heterogeneous Formats and Open-Ended Queries. arXiv:2509.21825
[15] Aymeric Roucher et al. 2025. `smolagents`: a smol library to build great agentic systems. https://github.com/huggingface/smolagents.
[16] Weiming Sheng, Jinlang Wang, Manuel Barros, et al. 2026. Building a Correct-by-Design Lakehouse: Data Contracts, Versioning, and Transactional Pipelines for Humans and Agents. arXiv:2602.02335
[17] Treeverse. [n. d.]. lakeFS: Git-like capabilities for your object storage. https://lakefs.io/.
[18] Zuozhi Wang, Yicong Huang, et al. 2024. Texera: A System for Collaborative and Interactive Data Analytics Using Workflows. *VLDB* (2024).