Demonstration of BobFlow: Human-Agent Collaboration Using Dataflows on Data Science Tasks

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 coding scripts, but still inherits the complexity and low-level details of programming code. In this paper, we demonstrate a novel ReAct-based system called BobFlow, which is built on the idea of using dataflow as an abstraction for human-agent collaboration. This abstraction is especially friendly for LLMs to do their internal reasoning for data science tasks. In the demo, we will show the benefits of BobFlow developed on top of Apache Texera (Incubating), 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 finishes a task with high accuracy and low cost.

CCS CONCEPTS

• **Information systems** → **Data analytics**; **Computing platforms**;
• **Software and its engineering** → **Data flow architectures**.

KEYWORDS

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

PVLDB Reference Format:
Jiadong Bai, Yicong Huang, and Chen Li. Demonstration of BobFlow: Human-Agent Collaboration Using Dataflows on Data Science Tasks. PVLDB, 19(12): 4718 - 4721, 2026.
doi:10.14778/3827998.3828105

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Texera/bobflow.

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 analysts use natural language to ask agents to explore data, generate plots, and summarize findings via LLM services such as ChatGPT and Claude. While convenient, its interface typically limits analysts to high-level supervision [3] with 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 interaction becomes hard to manage for complex, multi-step tasks. Although recent reasoning LLMs reveal intermediate chain-of-thought traces, such traces are still plain text that is hard to manage and verify. Recent systems (e.g., Databricks Genie Code [2]) shift toward a "Script ReAct" paradigm, where the analyst and the agent collaborate on executable code and notebooks. As shown in the top part of Figure 1, the analyst describes the task in natural language, and the agent generates executable scripts (often notebook cells) as the shared artifact for collaboration. By grounding interaction in executable artifacts, Script ReAct offers greater transparency and control than conversational analytics [12].

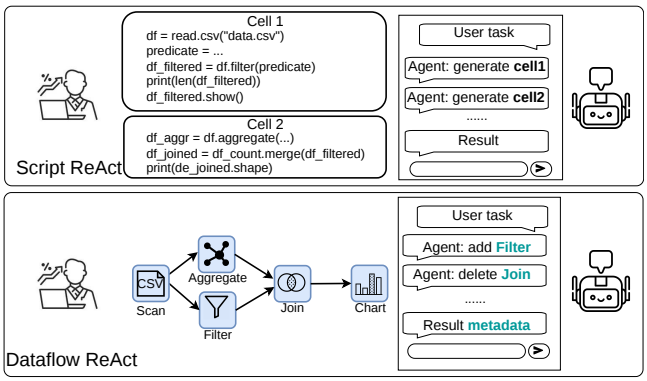

Figure 1: Script ReAct versus dataflow ReAct (BobFlow, this work) on data science tasks.

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 human analysts. They impose an entry barrier for non-programmers, and even experienced programmers spend significant effort reading code before they can understand, verify, or revise the intended analysis. (2) *Weak artifact traceability.* Although code can be version-controlled, execution output often resides in ephemeral artifacts such as in-memory states, temporary files, generated visualizations, and console logs, making it harder to track dependencies, compare intermediate results, and maintain provenance across iterations for both analysts and agents. (3) *Noisy agent context.* Code introduces low-level details such as syntax, variables, and control flow into the

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828105

agent context, increasing token cost, degrading grounding over long interactions, and raising the risk of hallucinated analysis steps.

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 had 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, 17]. Its explicit DAG of dependencies allows analysts to follow the structure of an analysis and trace different branches of the analysis. User studies further show that decomposing an analysis into modular operators with explicit dependencies helps analysts inspect and debug data analyses [4, 12, 15]. Relational DBMSs also expose execution plans through interfaces such as `EXPLAIN` in PostgreSQL. Such an abstraction can be especially friendly for LLMs to do their internal reasoning on data tasks.

Based on this insight, our idea, called *Dataflow ReAct*, is to use dataflow as the primary abstraction for human-agent collaboration in data science tasks (bottom of Figure 1). We present BobFlow, a system that uses interactive dataflows as the collaboration medium. On the user interface, analysts see and edit a visual dataflow iteratively constructed by the agent through a canvas and a chatbox. At the agent layer, the system captures context through the dataflow structure and operator-level semantics, and summarizes each operator's structured output (e.g., schemas and column statistics). At the execution layer, a dataflow engine runs the DAG of operators on a data lakehouse. At the core of BobFlow is an LLM-based agent called *Bob*, which understands an analyst's requests, constructs dataflows, presents them for inspection, and submits them to the engine, capturing dataflow-aware contexts and tracking their changes through governance versions. Across these layers, the dataflow serves as a shared abstraction for both the analyst and the agent *Bob*.

We have implemented BobFlow on top of the Apache Texera[1] system. In the demonstration, we show how BobFlow enables analysts to trace the agent's analytical process, inspect the rationale behind intermediate decisions, and intervene with effective feedback. Our preliminary benchmark shows that BobFlow achieves an accuracy of 79% for evaluated data science tasks, whereas a script-based ReAct baseline achieves 68%.

2 SYSTEM OVERVIEW

As shown in Figure 2, BobFlow consists of four layers: a graphical user interface (GUI), an agent called *Bob*, a dataflow engine, and a data lakehouse. The GUI includes a chatbox for analysts to communicate with *Bob* in natural language, and a canvas for visualizing a dataflow. The canvas exposes the dataflow's high-level structure, while optionally allowing analysts to inspect operator-level details such as code and property values (e.g., a threshold in a filter operator). Analysts can adjust the layout by dragging and dropping operators and can zoom in and out to navigate the DAG. With natural-language descriptions on operators, they can easily collaborate with the agent *Bob* even when the canvas contains many operators. The dataflow

[1]Currently under Apache incubation.

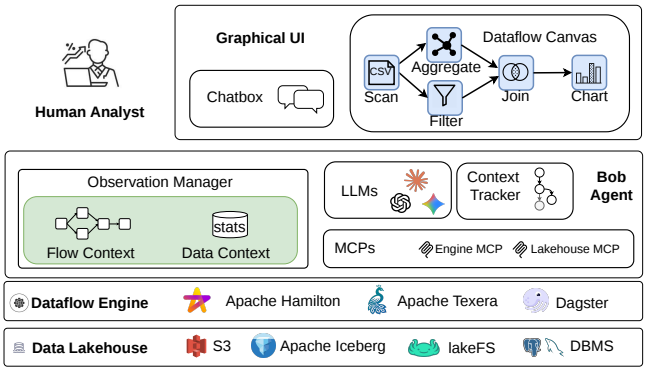

Figure 2: System architecture of BobFlow.

engine executes a dataflow, while the data lakehouse stores CSV files, generated artifacts, execution output, and runtime logs.

At the core, agent *Bob* coordinates reasoning, execution, and observing: it connects to LLMs and interacts with the engine and lakehouse through Model Context Protocols (MCPs). The engine MCP edits, inspects, and submits the dataflow for execution. The lakehouse MCP retrieves metadata and artifacts. Within the agent, an *Observation Manager* maintains two dataflow-oriented context abstractions, "Flow Context" and "Data Context," while a *Context Tracker* does version control of these contexts and other generated artifacts across iterations.

Human-Agent Collaboration with Dataflow ReAct Loop. Given a task from the chatbox, agent *Bob* starts the dataflow ReAct loop illustrated in Figure 4, taking multiple steps of *Think*, *Act*, and *Observe*. During thinking, the agent packages the task, the latest observations, and MCP instructions as a context and sends it to an LLM, which generates changes to the previous dataflow. The agent then acts to update the dataflow in the UI and submits the new *ExecutableDAG* to the engine. Once the execution completes, it collects observations on the new dataflow and the execution output, saves them in the data lakehouse, and repeats the loop until sufficient progress is made. Throughout the process, the analyst remains in the loop to monitor agent-driven changes, inspect execution results, interrupt execution, revise the dataflow, or provide feedback to redirect subsequent reasoning.

Observations Designed for Dataflow ReAct. The *Observe* step is designed not to feed raw code or low-level runtime states to *Bob*, but to extract two high-level, dataflow-oriented observations: *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. The Data Context summarizes properties of the data and generated artifacts, including schema, metadata, statistics, and distributions of intermediate and final results. In Figure 4, after step 2, the agent extracts the new operator *Chart* and its edge from *Join* into the Flow Context, and fetches the schema and statistics of the newly produced *Result(C)* into the Data Context in step 3. To keep observations lightweight, the Observation Manager stores pointers of the result tables in the lakehouse and, when assembling the context, retrieves data summaries subject to a predefined token cap rather than full tables.

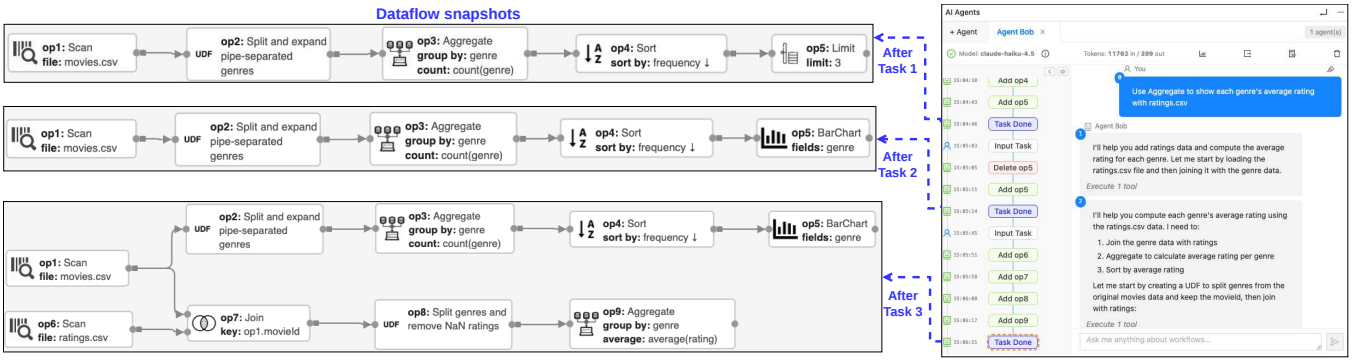

Figure 3: The BobFlow interface across three tasks: the canvas (left) shows the dataflow evolving in real time with operator-level semantics; the chatbox (right) displays the agent's reasoning and tracks context versions.

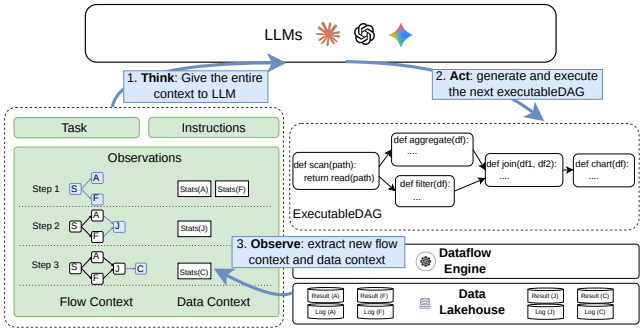

Figure 4: The "Think, Act, and Observe" loop for dataflow ReAct used in BobFlow.

Context Tracking with Governance Versions. To support iterative collaboration, the Context Tracker records a *governance version* each time *Bob* takes an action. Each version captures the dataflow together with a result pointer for each operator, while the computed results themselves are cached in the data lakehouse. When assembling the context for the LLM, *Bob* uses each operator's pointer to retrieve its result summary from the lakehouse, forming the Flow Context and Data Context. Because results are cached per operator, re-executing a dataflow recomputes only the operators that changed from the previous version and reuses the cached results of the rest. These versions give the analyst a complete history of *Bob*'s iterative actions. The analyst can check out any version to inspect its dataflow and results at that point with very low latency, since only the lightweight dataflow and pointers are restored. The analyst can also comment on a historical version to provide feedback, which prompts *Bob* to branch from it, restoring that version and starting subsequent ReAct steps from it.

3 DEMONSTRATION SCENARIOS

We will demonstrate BobFlow with an example where an analyst named Alice studies the correlation between movies and genres in two files, movies.csv and ratings.csv, sharing a movie-ID column (schemas and tasks simplified for illustration purposes).

3.1 Understanding Agent's Behavior through a Dataflow

Initially, Alice wants insights about the movie genres and sends the following task:

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

Alice follows agent *Bob*'s progress in real time on the canvas, which renders the latest dataflow after each ReAct iteration. Figure 3 shows the dataflow snapshot after *Bob* finishes this task. *Bob* begins by reading `movies.csv` with a scan operator to turn it into a table. As each movie record contains multiple genres, *Bob* uses a Python user-defined function (UDF) to split the nested genres into multiple records. It then adds an aggregate operator to count genre frequencies, a sort operator to rank them, and a limit operator to keep the top three. With the dataflow and its execution on the canvas, Alice easily understands the process and checks the result.

Next, Alice wants to visualize the genre distribution:

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

Agent *Bob* starts a new ReAct process. Alice can see that it resumes from task 1's context, with the version timeline showing continuity between the two tasks. The second snapshot in Figure 3 shows that *Bob* takes two ReAct steps. First, *Bob* understands that the analyst wants the full distribution of all genres rather than the top three, and removes the limit operator. Second, it adds a bar chart operator to display the genre distribution. With the help of *Bob*, Alice now has a better understanding of the genres, and sends a new task to relate ratings to genres:

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

The rest of Figure 3 shows the dataflow snapshot and the chatbox after *Bob* finishes task 3. *Bob* adds a scan operator to load `ratings.csv` and an operator to join it with the movies table on the column "movie ID." By checking the result's statistics, *Bob* identifies many records with "NaN" values. Thus, it connects another Python UDF to remove those records, followed by an aggregate operator to calculate the average rating per genre.

3.2 Inspecting Past Actions to Provide Feedback

Alice notices many "NaN" values in the final output, and wants to confirm whether they are produced by the join or come from the

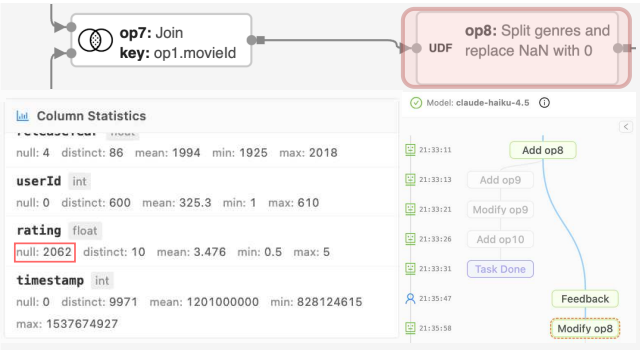

Figure 5: Alice inspects the join result's statistics, identifies several NaN ratings, and gives feedback to the UDF operator; *Bob* branches out to revise it.

original ratings file. Inspecting the join operator's results, she notices from the rating column's statistics that about 20% of records contain "NaN." By tracing downstream operators, she finds that the UDF operator drops them. Alice prefers to replace these values with zero so that genres' ratings are not inflated. She clicks on the past action that introduced the UDF operator, and sends the following feedback to *Bob*:

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

After receiving the feedback, *Bob* understands that it is a response on the past action: it switches the dataflow back to the version after that action, and utilizes the Context Tracker to branch out from it. It re-assembles a detached context containing the dataflow structure, execution results, and observations of that version. The new ReAct loop starts with the detached context combined with Alice's feedback, and the LLM returns an action that modifies the UDF's code to replace "NaN" values with 0. Alice sees a new branch from the past version, as shown in Figure 5; *Bob* builds future ReAct steps on this branch, while all previous actions are kept for Alice to investigate or provide feedback. After a few more ReAct steps, *Bob* finishes a new analysis dataflow, and Alice sees that the genres' ratings drop by 0.6.

3.3 Higher Accuracy and Lower Cost

The dataflow abstraction also improves the agent's performance: the extracted flow context and data context represent observations more concisely than Script ReAct, allowing the agent to retain useful context over longer interactions and focus on high-level reasoning. To evaluate this benefit, we compared BobFlow with a script-based ReAct baseline built on smolagents [16] using KramaBench [13], a data science benchmark that contains 104 tasks, with GPT-5-mini and GPT-5.2. As shown in Table 1, BobFlow achieved a higher accuracy at a lower cost on both models. For example, with GPT-5-mini, BobFlow improved the accuracy from 63.5% to 76.9% while lowering the per-task cost from $0.0204 to $0.0168.

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. As code serves as both the execution artifact and the agent's context, they suffer from noisy contexts filled with low-level details such as syntax, variables, and runtime values. 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 extra orchestration overhead. Both approaches provide limited or no interfaces for analysts, preventing effective collaboration on tasks requiring human feedback. LLM-powered systems such as SiriusBI [10] and DataChat [11] do offer interactive interfaces, but internally still generate scripts as context and expose them through chat-based or notebook-like interfaces. Our work differs by using dataflow as the primary abstraction in human-agent collaboration, offering the aforementioned benefits.

Table 1: Comparison on KramaBench [13]; cost is per task.

Agent	GPT-5-mini		GPT-5.2	
	Acc. (%)	Cost ($)	Acc. (%)	Cost ($)
Script Agent	63.5	0.0204	68.3	0.0964
Bob Agent	76.9	0.0168	79.1	0.0843
Improvements	+13.4	-17.6%	+10.8	-12.6%

ACKNOWLEDGMENTS

This work was supported by NSF award 2200274, NIH NIDDK award 2U24DK097771-11, and an SAP grant.

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] Databricks. 2026. Use Genie Code for data science. https://docs.databricks.com/aws/en/notebooks/ds-agent. Accessed: 2026-03-27.
[3] Ian Drosos, Advait Sarkar, Xiaotong Xu, Carina Negreanu, 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)*.
[4] Stephen N. Freund, Brooke Simon, Emery D. Berger, and Eunice Jun. 2025. Flowco: Mixed-Initiative Authoring of Reliable End-to-End Data Analyses via Dataflow Graphs and LLMs *(UIST '25)*.
[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 *(ICML '24)*.
[7] Sirui Hong, Yizhang Lin, et al. 2025. Data Interpreter: An LLM Agent for Data Science. In *ACL 2025*.
[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, Dylan Bacon, Junda Chen, 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, Gerardo Vitagliano, Ziyu Zhang, et al. 2025. KramaBench: A Benchmark for AI Systems on Data-to-Insight Pipelines over Data Lakes. arXiv:2506.06541.
[14] Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, et al. 2026. DS-STAR: Data Science Agent for Solving Diverse Tasks across Heterogeneous Formats and Open-Ended Queries. arXiv:2509.21825
[15] Shengquan Ni, Yicong Huang, Zuozhi Wang, and Chen Li. 2024. IcedTea: Efficient and Responsive Time-Travel Debugging in Dataflow Systems. *Proc. VLDB Endow.* 18, 3 (2024), 902–914.
[16] Aymeric Roucher et al. 2025. `smolagents`: a smol library to build great agentic systems. https://github.com/huggingface/smolagents. Accessed: 2026-07-06.
[17] Zuozhi Wang, Yicong Huang, Shengquan Ni, et al. 2024. Texera: A System for Collaborative and Interactive Data Analytics Using Workflows. *VLDB* (2024).