DA-Studio: An Agentic System for End-to-End Data Analysis

Yizhe Liu
Renmin University of China
liuyizhe2004@126.com

Shaolei Zhang*
Renmin University of China
zhangshaolei98@ruc.edu.cn

Ju Fan
Renmin University of China
fanj@ruc.edu.cn

ABSTRACT

Real-world data analysis is a multi-step process over heterogeneous inputs rather than merely producing a final answer. A practical system should autonomously organize multi-step workflows, execute generated code in a sandboxed environment, and remain inspectable through visible action traces and intermediate artifacts. Existing LLM-based analysis tools, however, often emphasize isolated subtasks, leaving limited support for complete execution-grounded workflows.

We present **DA-Studio (Data Analysis Studio)**, an interactive web-based demo system for autonomous, sandboxed, and inspectable end-to-end data analysis. DA-Studio integrates a structured analysis backend, a sandboxed workspace, and a browser interface for task setup, streamed traces, artifact preview, code editing and rerunning, and report export. Through iterative action generation, code execution, and feedback incorporation, it constructs executable analysis steps from raw files and natural-language requests while exposing intermediate results and artifacts throughout the process.

PVLDB Reference Format:
Yizhe Liu, Shaolei Zhang, and Ju Fan. DA-Studio: An Agentic System for End-to-End Data Analysis. PVLDB, 19(12): 4766 - 4769, 2026.
doi:10.14778/3827998.3828117
PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ruc-datalab/DeepAnalyze/tree/main/demo/chat_v2.

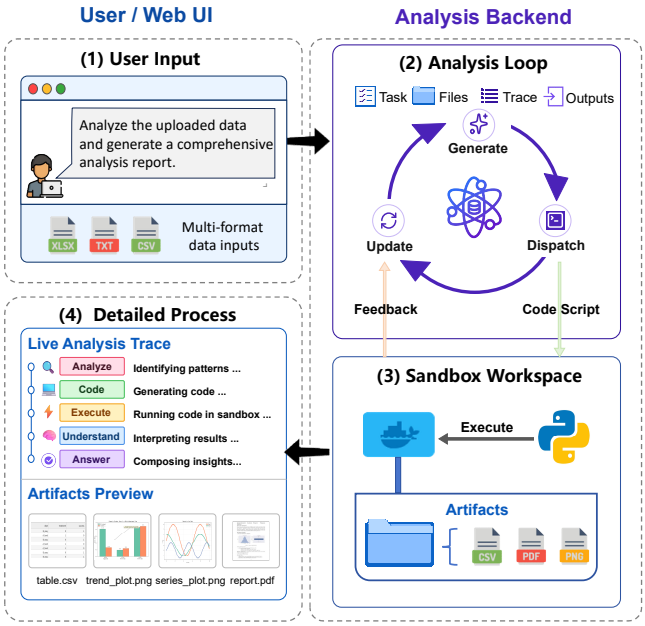

Figure 1: The backend converts user tasks and multi-format files into iterative action generation, sandboxed code execution, and feedback incorporation, while streaming traces and artifacts back to the web interface.

1 INTRODUCTION

Domain experts often need to analyze heterogeneous data, yet turning raw files into executable analysis steps and final deliverables still requires substantial manual effort. Recent LLM-based systems such as LIDA, Data Formulator 2, and nvAgent have improved natural-language-driven visualization authoring and exploratory analysis [1, 3, 5]. However, these systems mainly focus on visualization authoring or other isolated subtasks, leaving users to bridge the gap from raw inputs to final deliverables.

For end-to-end data analysis, a system should meet three requirements. First, it should autonomously orchestrate multi-step workflows over diverse file formats without requiring users to manually specify a pipeline. Second, it should execute generated code in a sandboxed environment over user data. Third, it should make the analysis process inspectable through visible action traces and intermediate artifacts.

Recent execution-grounded LLM systems have shown that models can iteratively generate code, execute it, and refine subsequent actions based on execution results [2, 4]. In particular, DeepAnalyze [6] is an agentic LLM trained for autonomous data science that emits structured analysis actions, but it is a model rather than a deployable system and therefore still needs a sandboxed execution backend, inspectable user interface and output controls for end users. To bring these capabilities into an end-to-end data analysis setting, we present **DA-Studio (Data Analysis Studio)**, an interactive web-based demo system for autonomous, sandboxed, and inspectable data analysis. DA-Studio combines an action-structured analysis backend, a sandboxed execution environment, and a user interface for streamed traces, artifact preview, code editing and rerunning, and Markdown/PDF report export. Through this interface, users can inspect not only the final results but also the process that produces them. This end-to-end visibility makes autonomous analysis easier to inspect, revise, and reuse.

Overview. Figure 1 gives a high-level view of how DA-Studio connects the User/Web UI with the Analysis Backend during an analysis session. (1) **User Input**: a user specifies an analysis request

*Corresponding author: Shaolei Zhang.

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828117

in natural language and uploads multi-format data files through the web interface. These inputs are submitted to the backend as the initial task and file context. (2) **Analysis Loop**: given the task and files, the model iteratively generates structured analysis actions, including planning, code generation, and result interpretation. When a complete <Code> action is produced, the backend extracts the script and sends it to the sandboxed workspace. The produced traces and execution outputs are then appended to the session context for later analysis steps. (3) **Sandbox Workspace**: the script is executed in an isolated session-scoped workspace. The sandbox returns execution feedback and registers generated artifacts, such as tables, figures, and reports, for later inspection and export. (4) **Detailed Process**: throughout the workflow, DA-Studio streams the live action trace, execution feedback, and generated artifacts back to the interface, allowing users to inspect intermediate results and intervene through preview, editing, rerunning, and export.

Demonstration Workflow. In the demonstration, users interact with the system by issuing natural-language analysis requests over heterogeneous inputs such as tables and documents. The interface exposes action traces, code, intermediate artifacts, and exported reports, allowing the audience to follow a complete execution-grounded analysis session from task setup to final deliverables. A demonstration video is available on YouTube.[1]

In summary, DA-Studio makes three contributions. First, it supports autonomous, execution-grounded multi-step analysis through an action-structured backend. Second, it provides sandboxed execution with session-scoped workspaces and artifact management. Third, it offers a user interface that unifies data upload, streamed traces, artifact preview, code revision, and report export.

2 SYSTEM ARCHITECTURE

To realize autonomous, sandboxed, and inspectable end-to-end data analysis, DA-Studio is organized as a five-layer architecture shown in Figure 2. We describe this architecture through three functional views that expose the system's key technical mechanisms: the Application Layer for inspectable interaction, the Model and Context Layers for autonomous multi-step analysis, and the Data and Environment Layers for sandboxed execution and artifact management.

As illustrated in Figure 2, these layers are linked by the flow of task requests, structured actions, execution feedback, and artifacts, forming a session-scoped workflow from task specification to sandboxed execution, artifact tracking, and user intervention.

2.1 Application Layer

To support inspectable and interactive analysis, the Application Layer manages user requests, streamed outputs, and workspace state rather than serving as a passive front end. Each user request is organized as a session-scoped task configuration that bundles the natural-language instruction, selected files, model/runtime parameters, and session identifier. The configuration is forwarded to the lower layers, while uploaded files are immediately registered in the session workspace managed by the Data Layer.

During execution, the Application Layer incrementally renders structured action blocks, code snippets, execution feedback, and final answers, while synchronizing the workspace tree, artifact preview pane, and code editor with the same session state. The interface maintains two synchronized views: the streamed trace shows the current analysis actions and execution feedback, while the workspace view exposes the files and intermediate artifacts materialized in the session, allowing users to inspect evolving outputs without waiting for the workflow to finish.

The Application Layer also supports post-execution intervention. Generated code blocks can be opened, edited, and rerun in the same workspace, and the final report together with the produced artifacts can be exported in Markdown/PDF form. Consequently, the Application Layer serves as both a visible interface for streamed outputs and an interactive entry point for code revision and rerunning.

2.2 Model and Context Layers

To support autonomous multi-step analysis without relying on manually predefined workflows, the Model and Context Layers use a unified action protocol to coordinate model outputs with session state and sandbox execution.

Model Layer. The Model Layer standardizes backend outputs into a unified action protocol for multi-step data analysis. Rather than following a manually predefined workflow, the model emits the next action on demand, choosing among planning, data inspection, code generation, result interpretation, and final answer generation. Following the action format used in DeepAnalyze [6], DA-Studio adopts tags such as <Analyze>, <Understand>, <Code>, and <Answer> to format model outputs as structured analysis actions, while an <Execute> block wraps the returned execution feedback, enabling the model to organize these actions and autonomously select the next step.

DA-Studio further adds a model-compatibility layer on top of this interface. When using DeepAnalyze, the backend can directly consume the model's native action blocks; when using general-purpose LLMs, it uses a prompt-constrained compatibility layer to enforce the same output format. In both cases, an output parser validates the structure and completeness of generated action blocks, and code is dispatched for execution only after a complete <Code> segment is finalized. As a result, heterogeneous models can be normalized into the same executable action interface, while code execution remains gated by explicit structural checks.

Context Layer. The Context Layer maintains the evolving state required by this protocol. Following a data-oriented context construction strategy similar in spirit to DeepAnalyze [6], it organizes the task instruction and lightweight file descriptors into a structured prompt, keeping the full file contents out of the context window. The model can then retrieve relevant data on demand through code execution and the returned results, avoiding treating heterogeneous raw files as static prompt text.

Operationally, the Context Layer maintains a rolling session state consisting of task instructions, prior action traces and execution outputs. When a completed <Code> block is produced, the corresponding script is extracted and dispatched to the Environment Layer. Returned program outputs are wrapped into a <Execute> block and reinjected into the context for the next round. As a result, the backend advances the analysis step by step, using execution results from each script to guide later model actions rather than producing a one-shot response.

[1] https://youtu.be/CnKEGfunLAI

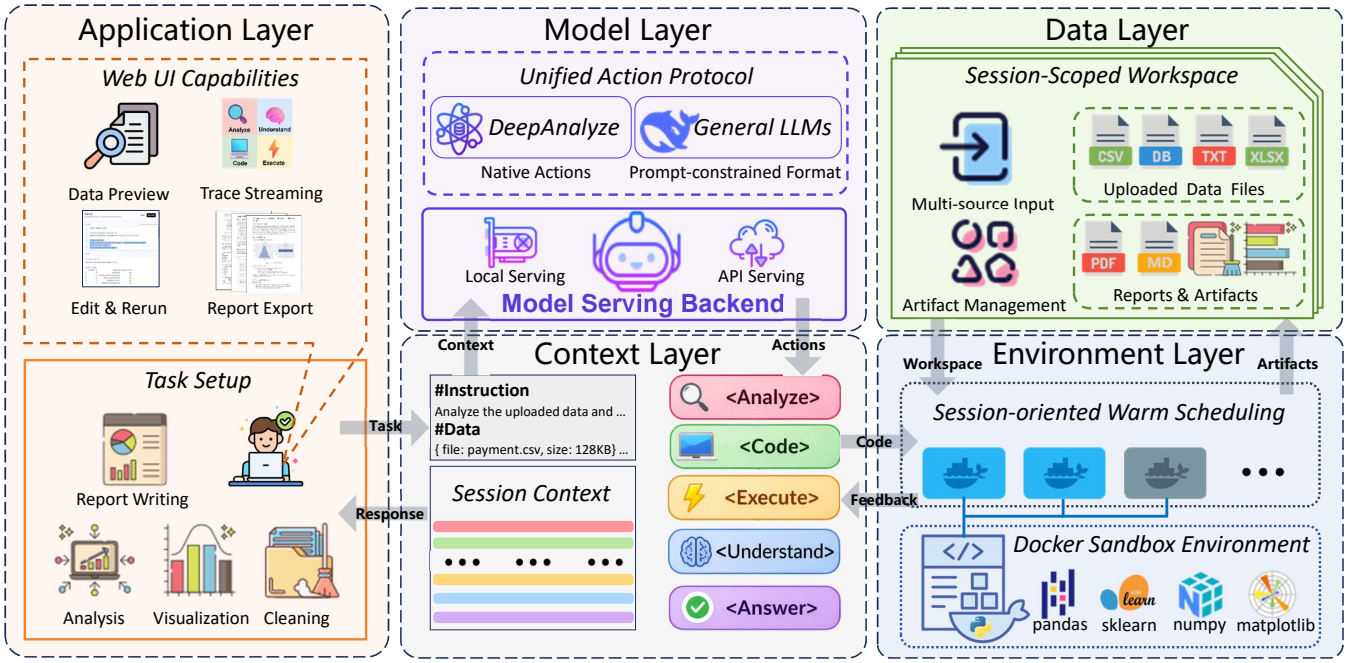

Figure 2: Five-layer architecture with three functional views. The Application Layer supports inspectable interaction, the Model and Context Layers enable autonomous multi-step analysis, and the Data and Environment Layers ensure sandboxed execution over user data.

2.3 Data and Environment Layers

To ensure sandboxed and resource-bounded execution over user data, the Data and Environment Layers provide isolated runtime support and workspace-level file management.

Data Layer. The Data Layer manages both uploaded inputs and generated outputs throughout the task through a session-scoped workspace abstraction. Each session is assigned an independent workspace root, and all file operations such as upload, preview, and cleanup are resolved against this root through path-safe APIs. This organization provides file-level isolation for executing the generated code.

Beyond raw storage, the Data Layer also serves as an artifact-management component. Newly generated outputs such as cleaned tables, plots, and summaries are registered in a dedicated generated namespace together with an artifact index, allowing the system to distinguish user-provided inputs from system-created derivatives. This indexing mechanism supports several user-facing capabilities, including incremental browsing of generated outputs during execution, bundled export of analysis results, and artifact references that can be reused when rerunning code and assembling the final report.

Environment Layer. The Environment Layer executes model-generated code inside a Docker-based sandbox with a stable Python data-analysis stack. Each <Code> block is materialized as a temporary script in the session workspace and executed in a container whose working directory is mounted to that workspace, confining file access to the current session and enabling controllable execution of generated scripts.

To reduce latency in iterative analysis, DA-Studio adopts *session-oriented warm scheduling*. The first execution request of a session triggers allocation of a dedicated container, and subsequent requests from the same session reuse the warm container; different sessions are mapped to different containers, and idle containers are reclaimed after timeout. Before and after each execution, the system snapshots the workspace to detect added or modified files, registers them as generated artifacts, and returns corresponding references to upper layers.

3 DEMONSTRATION SCENARIOS

Figure 3 shows a merchant-payment task with multi-format inputs and a natural-language request; the five highlighted regions correspond to the main stages of the workflow.

(1) Starting from task setup. As shown in the left panel of Figure 3, the workflow begins with a user uploading a merchant-payment dataset and specifying an analysis request in natural language. In this running example, the workspace contains data and supporting documents in multiple formats, including `payments.csv`, `fees.json`, and `manual.md`. The user then inputs a task instruction or chooses a preset prompt for common tasks such as exploratory analysis or visualization-oriented reporting. The interface additionally exposes model configuration options, including the system prompt, decoding temperature, and model selection.

(2) Following the analysis as it unfolds. After submission, the user's attention shifts to the central streaming panel, where DA-Studio reveals the ongoing analysis trace step by step. In this running example, the model first uses an <Analyze> block to plan the initial

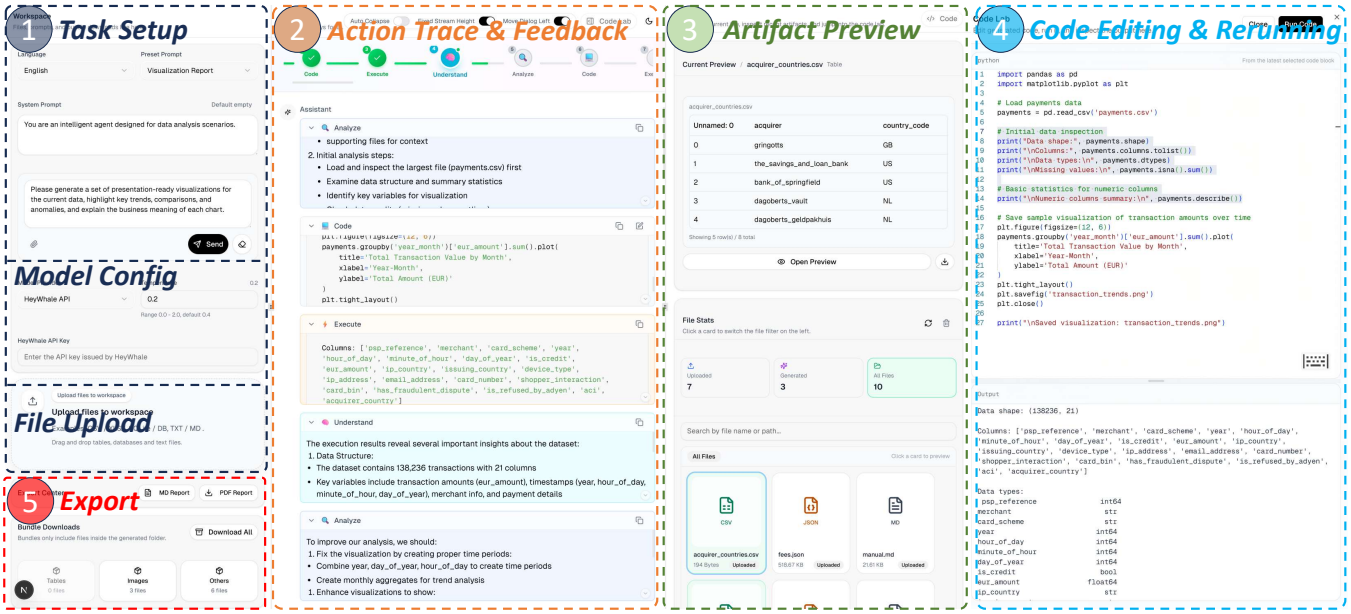

Figure 3: Screenshot of DA-Studio during a merchant-payment analysis demo. Region (1) supports task setup over heterogeneous files, (2) streams action traces and execution feedback, (3) previews intermediate artifacts, (4) enables code editing and rerunning, and (5) exports reports and bundled outputs.

steps, and then emits a <Code> block to generate code for reading and processing the relevant files. After receiving execution feedback through an <Execute> block, it enters an <Understand> block to interpret the data structure and contents, and then returns to <Analyze> to plan the next action. This iterative process is continuously streamed in the central panel of Figure 3, allowing users to follow how the analysis progresses from one step to the next.

(3) Inspecting intermediate artifacts. While the streamed trace explains what the system is doing, the right-side workspace in Figure 3 shows what the system has produced so far. Uploaded files remain visible next to newly generated artifacts such as cleaned tables, figures, and execution summaries. The user can click file cards in the right-side panel to preview either uploaded inputs or newly generated artifacts in real time, enabling verification of intermediate results without leaving the current session.

(4) Revising and rerunning generated code. Once the main interaction loop has produced a result, the user can continue working with the generated scripts through the integrated code editor. Instead of restarting the full analysis from scratch, the user may open a code block, make targeted changes by providing modification instructions in natural language or by editing the code directly, and rerun the script in the same sandboxed workspace.

(5) Exporting final results and analysis traces. The workflow ends in the export area highlighted in Figure 3. After reviewing the outputs, the user can export the final report in Markdown or PDF form and download the generated artifacts as a bundled archive. The report summarizes both the final findings and the analysis trace, while the bundled outputs preserve the tables, figures, and other files produced during execution.

ACKNOWLEDGMENTS

We thank all the anonymous reviewers for their insightful and valuable comments. This work was partially supported by the National Natural Science Foundation of China (Grant Nos. 62436010, 62441230), the Scientific Research Innovation Capability Support Project for Young Faculty (Grant No. SRICSPYF-ZY2025001) and Beijing Natural Science Foundation (L244010).

REFERENCES

[1] Victor Dibia. 2023. LIDA: A Tool for Automatic Generation of Grammar-Agnostic Visualizations and Infographics Using Large Language Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*. Association for Computational Linguistics, Toronto, Canada, 113–126. https://doi.org/10.18653/v1/2023.acl-demo.11

[2] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-Aided Language Models. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Vol. 202. PMLR, Honolulu, Hawaii, USA, 10764–10799. https://proceedings.mlr.press/v202/gao23f.html

[3] Geliang Ouyang, Jingyao Chen, Zhihe Nie, Yi Gui, Yao Wan, Hongyu Zhang, and Dongping Chen. 2025. nvAgent: Automated Data Visualization from Natural Language via Collaborative Agent Workflow. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Vienna, Austria, 19534–19567. https://doi.org/10.18653/v1/2025.acl-long.960

[4] 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. 2023. TaskWeaver: A Code-First Agent Framework. https://doi.org/10.48550/arXiv.2311.17541 arXiv:2311.17541 [cs.AI]

[5] Chenglong Wang, Bongshin Lee, Steven Drucker, Dan Marshall, and Jianfeng Gao. 2025. Data Formulator 2: Iterative Creation of Data Visualizations, with AI Transforming Data Along the Way. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. ACM, Yokohama, Japan, Article 677, 17 pages. https://doi.org/10.1145/3706598.3713296

[6] Shaolei Zhang, Ju Fan, Meihao Fan, Guoliang Li, and Xiaoyong Du. 2025. DeepAnalyze: Agentic Large Language Models for Autonomous Data Science. https://doi.org/10.48550/arXiv.2510.16872 arXiv:2510.16872 [cs.AI]