Co-Intelligence Visual Analytics: Verifiable Agent Workflows for Investigating Anomalous SaidIT Posts

Shibo Yang* Beihang University
Yuan Zhang† Beihang University
Kai Xu‡ University of Nottingham
Zipeng Liu§ Beihang University

Abstract

We propose a trace-based co-intelligence visual analytics framework and apply it to VAST Challenge 2026 Mini-Challenge 2. The framework records the analytical work of autonomous agents as an inspectable workflow trace comprising task specifications, intermediate artifacts, data transformations, visual evidence, and candidate findings. It enables analysts to guide the agent's analytical process, scrutinize the evidential basis of generated conclusions, and revise subsequent tasks. Applied to Mini-Challenge 2, the framework enabled the identification of a propagation chain spanning 7.8 days and comprising 191 events, which carried SwiftWren.txt to a SaidIT post, as well as two historical incidents sharing the same operational pattern. These findings motivated our proposal of a validation mechanism at the common final publication step. The workflow trace further supported human validation of the agent's analysis.

Index Terms: Co-intelligence, Visual Analytics, Provenance, VAST Challenge Mini-Challenge 2.

1 Introduction

VAST Challenge 2026 Mini-Challenge 2 asks participants to investigate anomalous SaidIT posts from heterogeneous synthetic records [4]. The analysis requires reconstruction of each posting sequence, its system-wide context and probable content origin, prior occurrences, and one intervention point. These tasks demand extensive integration of temporal and relational evidence, data transformation, and iterative analysis.

Unlike a conventional VAST Challenge workflow in which analysts conduct data preparation, visual exploration, and interpretation directly, we use AI systems for operational analysis while retaining human judgment. To support rigorous and verifiable visual analysis, our trace-based framework records each task's inputs, transformations, artifacts, visual evidence, and candidate claim. It enables analysts to inspect intermediate work, redirect erroneous analysis, and assess whether a result is adequately supported.

2 Data Processing and Framework Design

Our data processing was deliberately open-ended rather than prescribed as a fixed pipeline. In response to emerging evidence, analysts articulated processing requirements or analytical ideas, and a coding agent implemented the corresponding transformations, filtering, and comparisons with Python and command-line tools.

For example, analysts detected a potential mismatch between the time zone in the Challenge description and dataset timestamps. They asked the coding agent to locate the anomalous post and infer the offset by comparing the two temporal references. The agent transformed and filtered the records, reported the inferred offset for human confirmation, and the confirmed offset was then applied to subsequent temporal analyses.

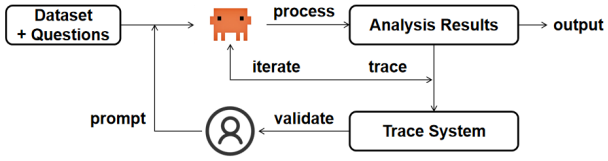

Figure 1: Trace-based framework for co-intelligence visual analytics. Analysts provide prompts and validate intermediate results; the coding agent processes the dataset and produces analysis results, while the trace system records and presents the agent workflow for iterative human guidance.

The framework provides a complete trace record and presentation of the collaborative analytical process. For each task, the trace captures its specification, relevant inputs, intermediate transformations, generated code and artifacts, visual evidence, and candidate findings; a directed acyclic graph (DAG) view links claims to the tasks, operations, and data artifacts that support them. Analysts inspect these intermediate nodes and evidence, issue corrective instructions when needed, and validate subsequent candidate views and findings. This iterative cycle retains human responsibility for the investigative agenda, interpretation, and validation while using the coding agent to accelerate data manipulation, provenance tracing, hypothesis generation, and preliminary visualization construction. We implemented the framework with Python, Matplotlib, Excel, D3 [2], React [3], Claude Code [1], and Codex.

3 System Overview

The AgentTrace Workbench comprises three coordinated regions (Fig. 2). On the left, a trace timeline summarizes the semantically defined steps in the agent workflow. Each step card records its order, type, and task description. This view gives analysts a concise account of the agent's progression and a direct entry point for selecting a step for inspection.

The center region presents a directed acyclic graph (DAG) derived from the agent workflow. Its visual organization is data-transformation-centric: because each analytical result must be derived from raw data, the graph follows the transformation lineage from source datasets through intermediate datasets to result datasets. It distinguishes dataset, code, report, and visualization nodes, and their connections make data-flow and analytical dependencies explicit. The view supports switching among analytical tasks and continuing the analysis, allowing analysts to inspect different task traces without losing the overall task context.

On the right, an inspector exposes detailed information for the selected node, including its objective, input and output files, data samples, and code logic. A reviewer agent generates the detailed descriptions and AI-assisted auxiliary charts used to facilitate interpretation of dataset nodes. For code nodes, the reviewer agent translates the implementation into `algorithm_purpose` and

*e-mail: zy2424122@buaa.edu.cn
†e-mail: zhangyuan3707@gmail.com
‡e-mail: kai.xu@nottingham.ac.uk
§e-mail: zipeng@buaa.edu.cn

`algorithm_logic`. At the completion of a stage, the system presents candidate `key_findings`, question answers, and associated visualization artifacts for human judgment and validation. These visual artifacts are themselves produced through the same co-intelligence process. Together, the three regions connect a concise task-level summary, a data-lineage view, and node-level evidence for verification and further analysis.

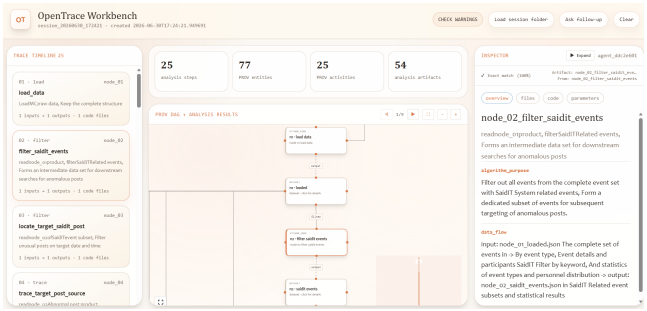

Figure 2: AgentTrace Workbench interface for inspecting and guiding AI-assisted analysis workflows.

4 Findings

4.1 Event Chain

The anomalous propagation chain is a multi-step file and task-propagation sequence centered on *`SwiftWren.txt`*. On **May 9, 2046**, **Emma Harbor** created this file, accessed *`SwiftWren_further_instructions.md`*, and queued subsequent reading tasks. The tasks were then relayed through agents associated with **Evelyn**, **Chloe**, **Mia**, **Levi**, **Owen**, and **Victoria** before reaching **John Windward**. On **May 17**, John used *`SwiftWren.txt`* as the **`content_source`** for a post to the **SaidIT** general forum; the post was deleted two seconds later.

The reconstructed chain contains 191 **SwiftWren**-related events, of which 186 are queued propagation events, and spans approximately 7.8 days. The system overview distinguishes direct-content posts from posts based on `.txt` files. Relative to the other observed posting chains, the **SwiftWren** chain is unusually long and crosses multiple personnel and organizational relationships. Thus, the anomaly is not only the terminal posting event, but also the extended relay of file-based instructions that preceded publication.

4.2 Content Provenance

`SwiftWren.txt` was a 30,615-byte text file created by **Emma Harbor** and referenced as the **`content_source`** of **John**'s **SaidIT** post. In challenge local time (**UTC−07:00**), **Emma** read *`meeting_notes.doc`* at **2046-05-09 07:02:00** and created *`SwiftWren.txt`* one second later. This temporal adjacency, combined with the subsequent instruction and task-propagation sequence, indicates that the file was likely extracted, organized, or transformed from the meeting notes before being propagated toward publication.

We examined the five meetings attended by **Emma** before the file was created. The strongest candidates were ***Executive Suite Sync: Work-Order Triage & SLA Tracking for Resident Messaging*** and ***Executive Suite: maintenance requests sync***. Their topics suggest that the intended material concerned property operations, such as maintenance requests, work-order triage, vendor dispatch, service-level tracking, resident messaging, or building announcements. This interpretation is a content hypothesis rather than a confirmed reconstruction.

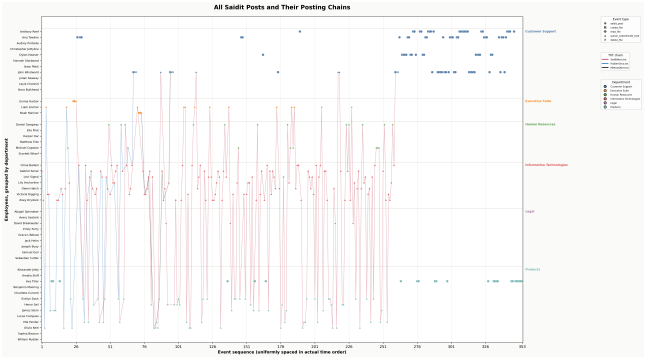

Figure 3: SaidIT posts and associated text-file-based posting chains.

4.3 Recurrence and Intervention

Historical analysis identified two earlier **SaidIT** posts by **John Windward**. The *`HiddenOrca.txt`* chain contains 42 related records; its post was made on **May 10, 2046** and deleted two seconds later. The *`MellowOtter.txt`* chain contains 15 related records; the file, created by **Noah Mariner**, was posted on **May 11** and likewise deleted after two seconds. Across all three cases, a similarly named `.txt` file is used as the **SaidIT** content source, a corresponding **Markdown** instruction file is created and read, and instructions are propagated through tasks before **John** publishes the content. The source and instruction files are subsequently deleted. Although the chains differ substantially in length, they share multiple participants and the same operational pattern, indicating that the behavior may recur under the same file-propagation and posting conditions.

The observed behavior is enabled by permissive agent access and communication structures: agents can access source files with few file-system protections, issue instructions to other agents without human validation, and use cross-department links through the **IT** hierarchy to form command loops that reinject instructions into the organization. Because the text-file-based chains ultimately converge at the final **SaidIT** submission, we propose a single validation gate at this point. The gate should confirm that the source file was read successfully and reject empty, corrupted, or garbled content, thereby limiting anomalous dissemination while preserving upstream trace evidence for subsequent investigation.

5 Conclusion

The trace-based framework provides an interface for the evolving division of labor in AI-assisted visual analytics. It enables analysts to inspect AI-mediated operations, intervene when intermediate results are inadequate, and preserve an auditable path from question to evidence to conclusion. In the VAST Challenge application, the framework exposed the multi-step SwiftWren mechanism, its two precedents, and a focused safeguard: source-content validation immediately before SaidIT submission.

References

[1] Anthropic. Claude code documentation. https://code.claude.com/docs/en/overview. Accessed: 2026-08-13. 1

[2] D3.js. Data-driven documents. https://d3js.org. Accessed: 2026-08-13. 1

[3] React. The library for web and native user interfaces. https://react.dev. Accessed: 2026-08-13. 1

[4] VAST Challenge. Vast challenge 2026. https://vast-challenge.github.io/2026/, 2026. Accessed: 2026-08-13. 1