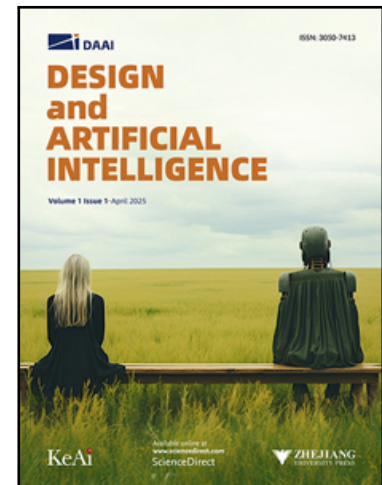


Journal Pre-proof

XAgen: Designing Explainability Support for Failure Detection and Correction in Multi-Agent Workflows

Xinru Wang, Zhuoran Lu, Shuai Ma, Ming Yin, Eunyee Koh, Mustafa Doga Dogan

PII: S3050-7413(26)00044-3
DOI: <https://doi.org/10.1016/j.daai.2026.100105>
Reference: DAAI 100105



To appear in: *Design and Artificial Intelligence*

Received date: 12 June 2026
Revised date: 27 August 2026
Accepted date: 17 September 2026

Please cite this article as: Xinru Wang, Zhuoran Lu, Shuai Ma, Ming Yin, Eunyee Koh, Mustafa Doga Dogan, XAgen: Designing Explainability Support for Failure Detection and Correction in Multi-Agent Workflows, *Design and Artificial Intelligence* (2026), doi: <https://doi.org/10.1016/j.daai.2026.100105>

This is a PDF of an article that has undergone enhancements after acceptance, such as the addition of a cover page and metadata, and formatting for readability. This version will undergo additional copyediting, typesetting and review before it is published in its final form. As such, this version is no longer the Accepted Manuscript, but it is not yet the definitive Version of Record; we are providing this early version to give early visibility of the article. Please note that Elsevier's sharing policy for the Published Journal Article applies to this version, see: <https://www.elsevier.com/about/policies-and-standards/sharing#4-published-journal-article>. Please also note that, during the production process, errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

Copyright © 2026 The Authors. Publishing services by Elsevier B.V. on behalf of Zhejiang University Press Co., Ltd. and KeAi Communications Co. Ltd.
This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>)

Highlights

XAgen: Designing Explainability Support for Failure Detection and Correction in Multi-Agent Workflows

Xinru Wang, Zhuoran Lu, Shuai Ma, Ming Yin, Eunyee Koh, Mustafa Doga Dogan

- XAgen is an explainability tool for debugging multi-agent workflow systems.
- Interviews with 12 practitioners reveal 5 challenges and 4 design goals.
- Design includes log visualization, expert feedback, and automated error checking.
- User evaluation shows strong preference over a state-of-the-art baseline tool.
- Design guidelines proposed for explainability in collaborative agentic systems.

XAgen: Designing Explainability Support for Failure Detection and Correction in Multi-Agent Workflows

Xinru Wang^{a,*,1}, Zhuoran Lu^b, Shuai Ma^c, Ming Yin^b, Eunye Koh^d and Mustafa Doga Dogan^e

^aSingapore-MIT Alliance for Research and Technology, Singapore

^bPurdue University, West Lafayette, Indiana, USA

^cInstitute of Software, Chinese Academy of Sciences, Beijing, China

^dAdobe Research, San Jose, California, USA

^eAdobe Research, Basel, Switzerland

ARTICLE INFO

Keywords:

multi-agent systems
explainable AI
interactive debugging
log visualization
user study

ABSTRACT

The rapid adoption of LLM-powered multi-agent systems has brought users with diverse AI expertise—from software developers to domain experts—into the process of building and maintaining complex agentic workflows. Yet when these distributed systems fail, their non-deterministic and opaque nature makes it difficult for users to observe, attribute, and correct errors. To understand practitioners' needs, we conducted formative interviews with 12 practitioners spanning AI developers who build agentic systems to domain experts who utilize them, surfacing five key challenge themes and four design goals for explainability tools. Building on these insights, we designed XAgen, an explainability interface that supports failure detection and correction through three capabilities: log visualization for glanceable workflow understanding, human-in-the-loop feedback to capture expert judgment, and automatic error identification via an LLM-as-a-judge, which reached 95% accuracy against author-verified labels. In a usability evaluation with 8 participants, XAgen was rated significantly higher than a state-of-the-art observability baseline across all four dimensions—understanding, usability, helpfulness, and satisfaction—with median scores of 6–7 out of 7 (all $p < 0.05$). Log visualization received the highest helpfulness rating (median = 5/5, $p < 0.01$), followed by automatic error identification (median = 4.5/5, $p < 0.05$), while human-in-the-loop output feedback was rated positively but only reached marginal significance (median = 4/5, $p = 0.07$). From these findings we distill four design guidelines for explainable agentic AI. Our work also highlights open challenges for context-aware interactive debugging of multi-agent workflows.

1. Introduction

Large Language Models (LLMs) have accelerated the democratization of multi-agent systems, where collections of specialized AI agents collaborate to solve complex tasks. These systems are increasingly applied in a wide range of domains, from website optimization [15, 31], co-creation [25, 9], to content editing [47, 33]. Unlike traditional monolithic AI models, multi-agent systems allow for modular and flexible orchestration of agents, each with distinct roles, goals, and capabilities [14, 12]. This flexibility makes multi-agent systems appealing to both practitioners and organizations who wish to tailor intelligent workflows to their needs.

As adoption grows, users with diverse AI expertise—from professional software developers to domain experts such as designers and sales analysts—are now building and refining their own agentic workflows [8, 7]. For these users, a central challenge to effectively optimize multi-agent workflows is to *identify and correct failure cases* over time: understanding when something has gone wrong and how to intervene.

Today, a common way to inspect a multi-agent workflow is by reading raw execution logs. These logs, often displayed in terminal windows, contain detailed traces of agent activities, communications, and tool usage. In addition, a growing set of developer-oriented dashboards, such as AgentOps [1] and LangTrace [23], and LangFuse [34], provide structured visualizations of features like call graphs, error reports, and execution traces. However, these tools require

*Corresponding author

✉ xinru.wang@smart.mit.edu (X. Wang); zhuoranlu33@gmail.com (Z. Lu); mashuai@iscas.ac.cn (S. Ma); mingyin@purdue.edu (M. Yin); eunye@adobe.com (E. Koh); doga@adobe.com (M.D. Dogan)
ORCID(s): 0000-0002-0213-6425 (X. Wang); 0000-0002-1079-2043 (Z. Lu); 0000-0002-7658-292X (S. Ma); 0000-0002-7364-139X (M. Yin); 0000-0003-2091-5972 (E. Koh); 0000-0003-3983-1955 (M.D. Dogan)

¹Work done during an internship at Adobe Research.

XAgent: Designing Explainability Support for Failure Detection and Correction in Multi-Agent Workflows

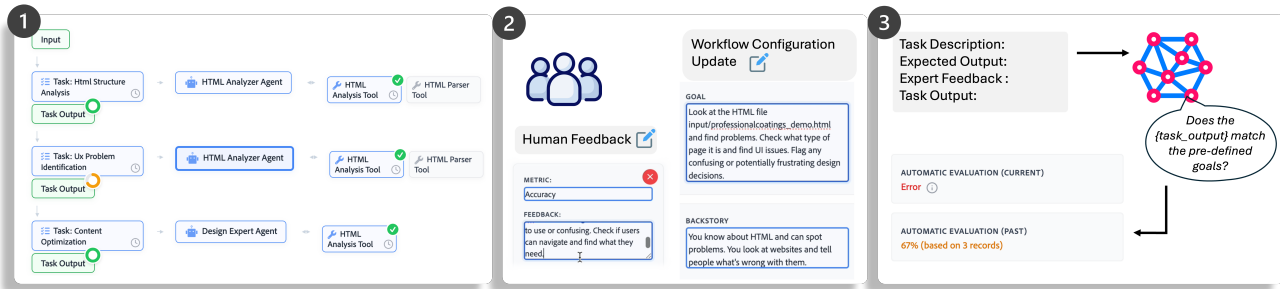


Figure 1: We introduce XAgent, an explainability tool that helps users with diverse AI expertise understand, detect, and correct failures in multi-agent workflows. XAgent integrates three core capabilities: (1) log visualization, which parses complex raw terminal logs into an interactive flowchart of agents, tasks, and tools, with each component in the flowchart activated step by step in accordance with the logs; (2) human-in-the-loop feedback, which enables users to review outputs, provide feedback, and iteratively update agent and task configurations directly in the interface before re-running the workflow to test improvements; and (3) automatic error identification, which uses an LLM-as-a-judge to evaluate task outcomes against predefined goals and any collected user feedback, with the historical success rates shown as a ring indicator on the node, and detailed scores and rationales displayed in the details panel.

substantial technical expertise and are primarily designed for AI developers. Non-technical users, such as domain experts who increasingly collaborate in building multi-agent systems, may need tools that allow them to more quickly and clearly comprehend system behavior and contribute meaningful corrections.

We argue that making multi-agent workflows more usable requires *importing and extending ideas from Explainable AI (XAI)*. Traditionally, XAI focuses on making model predictions more interpretable to human users for effective error detection and human intervention, often through approximations [26], counterfactuals [36], or rationales [44]. However, unlike traditional single-model systems, failures in multi-agent systems are often distributed and non-obvious. For instance, a content-generation workflow may produce an incoherent final output not because a single agent malfunctioned, but because an upstream summarization agent omitted critical context, which then propagated through downstream rewriting and formatting agents. Rather than explaining a single decision boundary, explaining multi-agent systems involves tracing *dynamic system-level behaviors between multiple agents and tasks*. This shift opens new opportunities for applying explainability concepts to workflows, interactions, and system-level behaviors. In this research, we asked: *What would it mean to design an explainability interface for multi-agent systems that helps users identify and correct failures, while supporting users with varying levels of technical background?* Without tools that bridge this gap, multi-agent systems risk becoming opaque “black boxes” that frustrate both technical and non-technical users, and undermining their potential as collaborative systems.

To inform our research, we first conducted semi-structured interviews with 12 practitioners who actively work with multi-agent systems in real-world settings, spanning *AI developers who build agentic systems* to *domain experts who utilize agent workflows* in practice. Across interviews, participants consistently highlighted pressing problems in debugging and understanding multi-agent systems. Developers noted the difficulty of tracing errors, identifying accountable agents, and comparing runs over time in hundreds of lines of logs. Non-technical domain experts expressed frustration at being unable to understand what agents were doing or how outputs were generated, let alone contributing meaningful feedback despite possessing valuable domain knowledge. Participants also emphasized the absence of reliable evaluation metrics that could indicate whether workflows were improving after configuration changes.

Using these findings, we designed XAgent, an explainability tool that helps users with varying levels of AI expertise identify and correct failure cases in multi-agent workflows. XAgent was developed around three representative features, directly motivated by the challenges and needs expressed in the formative interviews:

- **Log visualization for glanceable workflow understanding.** Rather than requiring users to parse raw logs, XAgent transforms messy execution traces into clean, interactive flowcharts. Each task, agent, and tool is represented as a block that allowing users to see at a glance how workflows progress and how components interconnect.

- **Human-in-the-loop feedback to capture expert judgment.** XAgent provides mechanisms for non-technical domain experts to review outputs and give feedback at any step, which can then be used to update agent or task configurations.
- **Automatic error identification with an LLM as a judge.** XAgent integrates an LLM-as-a-Judge component that evaluates task outcomes against predefined goals and accumulated human feedback. This feature automatically surfaces potential errors to guide users toward points of intervention.

Central to XAgent is that these three capabilities form a closed loop rather than a set of parallel functions. Expert feedback conditions the automatic judge, the judge's verdicts are displayed on the workflow structure users navigate, and corrections are made and re-tested in the same view. This integration allows users without developer skills to move from a flagged output to a tested fix within a single workspace.

We evaluated XAgent through a technical evaluation of its LLM judge against ground-truth labels and a usability study with 8 participants. Participants completed two multi-agent system improvement tasks using both XAgent and a baseline consisting of logs plus AgentOps visualization. Our study results show that participants perceived XAgent as better supporting them in locating failures, attributing them to specific agents or steps, and iteratively improving configurations. Participants also highlighted specific strengths of XAgent's features, while suggesting future improvements such as richer intermediate visualizations.

Taken together, our work makes the two key contributions listed below. We hope this work informs the design of explainable multi-agent systems that enable broader participation in building and refining agentic workflows.

- **Design guidelines for making multi-agent workflows explainable.** We elicit design goals for explainability tools from practitioners, and then, having built and evaluated a system against them, distill four design guidelines that demonstrate how concepts from explainable AI can be extended to multi-agent workflows to address the needs of users with diverse AI expertise, spanning both technical and non-technical roles. Unlike existing observability platforms, which offer evaluation and feedback as siloed, developer-facing pipelines, our guidelines call for a closed-loop integration in which failure detection, attribution, and correction are supported within a single explainability interface.
- **XAgent as a system for error detection and correction in multi-agent workflows.** We demonstrate and evaluate how its features—log visualization for failure detection, automatic error identification that localizes failures to individual tasks and agents, and human-in-the-loop feedback that is injected directly into the LLM judge's evaluation context—work together as a closed loop to support users in identifying and correcting failures in multi-agent systems.

2. Related Work

In this section, we review literature on advances in XAI and recent efforts to make multi-agent systems more explainable and observable.

2.1. Explainable AI for LLMs and Multi-Agent Systems

The concept of intelligent agents dates back to the 1990s, when AI research began shifting toward systems that perceive their environment, make decisions, and act autonomously to achieve goals [45]. In recent years, large language models (LLMs) have demonstrated remarkable capabilities not only in text generation and understanding but also in complex decision-making and multimodal tasks [28]. With their ability to call tools, store states, and reason step by step, LLMs can now function effectively as agents [14, 47]. For more complex problems, tasks are often decomposed and distributed across multiple specialized agents, each defined by a distinct role, prompt, and toolset, collaborating through shared memory or logging systems [14, 12].

The black-box nature of emerging AI models can lead to misconceptions [40, 41], bias [39], reduced trust and task success [38, 43], and miscommunication between humans and AI. This has motivated both practitioners and researchers to pursue XAI approaches that help human users understand the internal reasoning of AI models. Traditional XAI methods are often categorized as model-specific or model-agnostic. Model-specific techniques include designing inherently interpretable models such as rule-based systems and generalized additive models [19, 21]. Model-agnostic approaches, by contrast, include feature attribution [30, 26], example-based explanations such as prototypes [20], and counterfactual explanations [36]. Importantly, the goals of explainability span multiple dimensions: beyond

increasing transparency, explanations can support human understanding, error detection, enable human intervention and correction, and ultimately foster more effective human–AI collaboration [40, 41].

The emergence of LLMs poses unique challenges for traditional XAI, due to their massive training data, emergent capabilities, and non-deterministic outputs [6]. In response, researchers have explored strategies ranging from probing neuron activations [35] to analyzing cognitive representations in LLMs [52]. In addition, “self-reasoning” methods such as chain-of-thought [44], ReAct [48], and multimodal reasoning [51] make model outputs more interpretable while also guiding models’ reasoning processes toward better results.

While early work has pioneered XAI techniques for individual LLMs, explainability remains an open challenge for multi-agent systems. Existing XAI approaches typically assume a single-model setup. In contrast, multi-agent systems are inherently more complex: multiple agents perform diverse roles, trigger tool use, exchange messages, and coordinate over time. These interactions generate overwhelming amounts of text, which makes it difficult for a single human user (e.g., a developer) to evaluate the factuality and quality of results [11]. Consequently, explanation techniques designed for single models often fall short in providing the visibility needed for distributed workflows.

2.2. Observability and Visual Analytics Tools for Multi-Agent Systems

In software engineering and increasingly in agentic AI systems, the term *observability* refers to debugging tools that enable developers to monitor and infer the internal state of a system during operation from external signals such as logs, traces, and metrics. In the context of AI agents, observability supports performance monitoring (e.g., latency, resource usage, response time), error diagnosis and troubleshooting, and understanding agent decisions and interactions, which helps improve reliability and robustness.

Several frameworks have emerged to support multi-agent development, including CrewAI [3], AutoGen [27], and LangGraph [22], which streamline the orchestration of agent workflows. AutoGen also provides *AutoGen Studio* [5], a no-code interface for agent construction. Open standards such as the Model Context Protocol (MCP) [29] and the Agent2Agent (A2A) protocol [13] have also been proposed to enable interoperability across agent frameworks, toolchains, and infrastructure resources.

Researchers have used *visualization* techniques for complex LLM interactions to aid human understanding. While some visual toolkits support prompt engineering for single LLMs (e.g., [2, 18]), visualization of multi-agent behavior has also emerged [4]. Commercial observability platforms such as AgentOps [1], LangTrace [23], and LangFuse [34] provide visualization dashboards for inspecting agent behavior, LLM calls, event sequences, and errors. Recent work on interactive debugging tools [11, 49, 50, 16] further extends this by allowing users to step through agent conversations, edit agent behavior on the fly, and visualize multi-agent threads and state updates. In addition, Yan et al. [46] provides a step-through visualization of the exploration path of code navigation agents, which complements existing output-centric dashboards.

However, these platforms are designed primarily to record and analyze past executions. Some also support human annotation and automatic evaluation within their trace interfaces; Langfuse, for example, allows users to annotate traces and attach evaluation scores. Such annotations become scores or dataset entries for later evaluation, while changes to agent or task configurations are made outside the observability interface. Inspection, evaluation, and correction therefore remain separate parts of the debugging workflow rather than a continuous interaction within the same workspace.

These tools are also geared primarily towards developers. Other user groups, including designers, product managers, and domain experts who also need to identify and correct issues in multi-agent systems, may require different levels of transparency and support [32, 10]. We argue that extending XAI to multi-agent systems is crucial for helping such users understand system behavior and correct issues. A user-centered design approach to explanation in multi-agent workflows is therefore needed to build systems that align with the mental models and goals of users with diverse AI expertise levels.

3. Formative Interviews with Practitioners

3.1. Background on Multi-Agent Frameworks and Observability Tools

To provide background for our work, we first experimented with several commonly-used agentic AI frameworks to understand what methods currently support debugging practices of users developing their own multi-agent systems.

Agentic AI frameworks are building blocks that allow users to create and manage AI agents in a more scalable way rather than developing agents from scratch. These frameworks typically provide: (1) predefined architectures that

specify the structure, capabilities, and communication protocols of agents; (2) task management systems to coordinate workflows; (3) integration tools for function calling; and (4) monitoring utilities to track system performance. Several frameworks have gained popularity in the community, such as CrewAI [3], AutoGen [27], and LangGraph [22]:

- *CrewAI* is an open-source framework that organizes agents into role-based “crews”. Each agent is assigned a role, goal, and backstory, described in natural language. Each task is defined with responsibilities and expected outputs. A “crew” defines how agents and tasks are coordinated in a workflow. Workflows may be sequential or hierarchical under a manager agent.
- *AutoGen*, an open-source framework by Microsoft, employs message-driven chat flows between agents and supports development of network of agents with asynchronous multi-agent messaging. AutoGen also provides *AutoGen Studio* [5], a no-code interface for agent construction.
- *LangGraph*, part of the *LangChain* ecosystem, applies a directed-graph architecture, where agents are represented as nodes, and their transitions are modeled as edges. It supports cyclical, conditional, and nonlinear workflows.

To inspect the behavior of multi-agent systems, most frameworks provide detailed debugging logs of workflows and agent communications. However, because of the complex interactions and non-deterministic behavior of LLMs, AI agents are difficult to troubleshoot and monitor without proper support. Beyond these frameworks, several commercial *observability* tools have emerged, such as AgentOps [1], Langfuse [34] and Langtrace [23]. These developer-facing platforms address these challenges by providing visualization into execution traces and offer technical metrics such as time, tokens used, and error rates.

We evaluated and tested several AI observability tools to understand their pros and cons. We ran workflows in the codebase to generate traces, then opened the tool’s UI and inspect the traces, focusing on clarity, ease of navigation, and how well key details like errors are highlighted. Our evaluation considered four aspects: (1) integration with LLM and agent frameworks, (2) breadth of supported features, (3) interface intuitiveness, and (4) support for real-time monitoring.

- *AgentOps* is a Python SDK for monitoring AI agents, tracking LLM costs, and benchmarking. It integrates with frameworks like CrewAI, Langchain, and Autogen. Features include visual event tracking, time travel debugging, error and log tracking, prompt injection auditing, and LLM cost tracking. The platform offers an intuitive interface and real-time monitoring for agent performance.
- *Langfuse* is an open-source platform offering observability, metrics, evaluations, and prompt management. It integrates with frameworks like LangChain, but lacks native support for higher-level agentic frameworks (e.g., CrewAI). The interface facilitates collaborative debugging and real-time evaluation, but its flat session-based trace structure is better suited for chatbot workflows than complex multi-agent processes.
- *Langtrace* offers support for frameworks such as CrewAI and LangChain. Features include tracing, metrics, evaluations, prompt version control, and a playground. The interface includes dashboards for token usage, cost, and latency, enabling quick issue identification.

3.2. Interviews with Practitioners

While our initial exploration involved experimenting with the above-mentioned frameworks ourselves, we needed a deeper understanding of how real-world practitioners—especially in industry contexts where technical and non-technical users collaborate—build, understand, and improve multi-agent systems in their daily work, and the challenges they encounter with existing debugging methods.

We recruited 12 practitioners from our multinational organization, including 7 developers, 1 engineering manager, 2 domain experts, and 2 research scientists. The group consisted of 6 women and 6 men, with an average age of 35. Table 1 summarizes their backgrounds. All participants had prior experience experimenting with the aforementioned frameworks, though they relied on different ones in their day-to-day work. Most (10/12) primarily used CrewAI for development tasks, while the 2 research scientists focused on building their own custom agentic frameworks.

Each participant took part in a one-hour semi-structured interview. The interviews focused on uncovering pain points in agentic AI workflows and identifying mismatches between existing tools and user needs, particularly in human-agent co-creation scenarios and iterative debugging. We emailed interview questions in advance to prepare participants and asked them to bring a concrete example from a recent project. During the session, they walked us through how they used agentic AI in that context and reflected on questions such as:

Table 1

Formative study participants. Experience is reported in years. P2 and P5 are domain experts.

ID	Gender	Role	Work context	Framework	Experience (years)		
					Coding	AI	Agentic AI
P1	F	Machine learning engineer	Web development	CrewAI	10	7	<1
P2	F	Engineering/marketing manager	Web development	CrewAI	10	1.5	0
P3	M	Engineering manager	Web development	CrewAI	18	5	1
P4	M	Software development engineer	Web development	CrewAI	8	7	<1
P5	F	Product growth engineer	Web development	CrewAI	6	1.5	<1
P6	M	Research scientist	AI research	Custom	13	10	<1
P7	M	Research scientist	AI research	Custom	24	9	3
P8	F	Full-stack engineer	Web development	CrewAI	16	14	<1
P9	F	Software development engineer	Web development	CrewAI	10	8	<1
P10	F	Software development engineer	Web development	CrewAI	23	<1	<1
P11	M	Software development engineer	Web development	CrewAI	30	25	<1
P12	M	Software development engineer	Web development	CrewAI	15	10	<1

- What agentic AI tools (e.g., CrewAI, Autogen) have you used? What challenges have you faced in understanding what agents are doing or contributing?
- What observability tools (if any) do you currently use to track or debug agents?
 - How is your experience with using these observability tools?
 - What types of features or visualizations would help you better understand agent behavior/logs? Are there specific signals or forms of explanation that would improve usability or build trust?
 - What would your ideal agent observability and debugging experience look like?

All interviews were recorded and transcribed. We then conducted a thematic analysis of the transcripts and identified two sets of themes: (1) challenges in debugging and understanding agents, and (2) desired design features of explainability tools. These findings are detailed below.

3.2.1. Challenges in Understanding AI Agents

Our thematic analysis revealed five key challenges practitioners encounter when trying to understand and refine agentic AI workflows through execution logs and when engaging with existing observability tools. The five themes span different levels of the debugging experience: the first three (C1–C3) are interface-level explainability gaps, whereas the remaining two (C4–C5) show inherent limitations of LLM-based agents and organizational constraints.

C1. Interface and Usability Challenges. A recurring challenge practitioners highlighted was the difficulty of using current interfaces and logs effectively. First, all participants including domain experts described a **steep learning curve and onboarding difficulties**. For instance, P5 described “a very steep learning curve and a lot of frustration just to get it working in my local environ”, while P11 estimated that “somebody will have to spend one week to install, to onboard”. P8 similarly reflected on the manual effort required to explore libraries and codebases before being able to meaningfully use the system.

Second, participants pointed to **insufficient logging information**, which made debugging cumbersome. For example, P9 noted that “we have to manually add print statements in the code to see what’s happening. If some logs are missing, we still have to go back and add them ourselves when debugging.”

Finally, practitioners emphasized the **difficulty of managing large numbers of agents**. While a few logs may be manageable, scaling up quickly becomes overwhelming. As P7 described: “Right now, I run maybe four servers and can check different terminals to see where the error is. But if I had 50 agents, I cannot do this ... what is the right way to look at these errors?”

C2. Code and Execution Pipeline Complexity. Participants emphasized that challenges extend beyond individual agents to the end-to-end execution pipeline. Both developers and domain experts without a strong technical background noted that once a workflow begins, the **agent pipeline cannot be interrupted**, stressing the **need to track individual**

agent contributions. As P1 explained, *“With so many end-to-end tasks and agents, one flow takes too much time to finish before I can see the result and go back to improve—sometimes I wait ten minutes just to check a fix.”* When describing a three-agent loop, P7 asked, *“Which agent should get what credit for this particular task? What is the best way to do credit attribution? We still don’t know the answer.”* Participants also stressed the need for intermediate artifacts to localize errors. As P8 noted, *“Without appropriate logging and intermediate outputs, it’s hard to know which agent failed. Each agent ‘returns’ something, but if I expect JSON and get a weird format even once, the final output breaks. You then have to look closely to tell whether it’s a code issue, an agent-performance issue, or just an intermittent problem.”*

Participants also reported **issues stemming from the broader code pipeline rather than the agents themselves.** P5 described waiting an hour because the UI appeared “running,” while the terminal had already thrown an error: *“The errors weren’t giving me enough information about when they happened. . . as a non-technical person, it was difficult to understand what was going on behind the scenes.”* P8 and P9 pointed to mismatches between method names and actual functionality, fragile multi-step toolchains that complicate error localization, and the cognitive burden of remembering multiple cache layers.

C3. Output Evaluation and Performance Tracking. Participants also described challenges in evaluating outputs and tracking the performance of agentic workflows. One difficulty was **tracking performance across the workflow over time.** As P3 noted, *“with time it’s hard to get all the data and keep them reliable. . . the website changes, everything around changes, so it’s hard to crystallize it.”*

Another issue was the **lack of reliable metrics and the dependence on domain expertise to interpret output quality.** P2 explained that *“it’s very hard to achieve meaningful results using general-purpose LLMs. We should train our agents on domain-specific data. . . in our work we do have domain experts like SEO strategists, and they can manually evaluate results. But we don’t have an automated system for that.”* This reliance on human expertise highlighted the absence of systematic ways to validate agent outputs. P3 further pointed out the lack of short-term proxies for long-term outcomes, which makes it difficult to quickly assess whether an agent’s immediate actions would ultimately succeed.

While the first three themes point to gaps that a better explainability interface can directly address, the remaining two describe constraints that originate outside the interface layer: the probabilistic nature of LLMs and the organizational environment. We report them because they shape what users can realistically expect from any debugging tool, and because an explainability tool should help users recognize and localize their consequences even when it cannot eliminate their causes.

C4. Model Behavior and LLM-Specific Limitations. Another recurring challenge arose from the behavior of LLMs themselves and their inherent limitations. Participants mentioned the issue of **non-deterministic execution flows**, where repeated runs of the same workflow produced inconsistent outputs. For example, P10 explained, *“from my experience, when I get two generic results, just rerunning helps. . . maybe one in five runs it’s able to use MCP tools.”* Similarly, P8 described: *“the most troublesome part is that the agent’s outputs are not deterministic—sometimes you get a good format, sometimes nothing useful at all, and it’s impossible to reproduce. That’s the most frustrating part.”* P6 added that such nondeterminism is inherent to the model’s probabilistic nature.

Participants also highlighted **hallucinations and incorrect outputs** as a common barrier. As P7 put it, *“LLMs can hallucinate. . . they might produce an answer, but it may not be the best one, or even the right one.”* P11 observed that agents often continue executing even when outputs are clearly wrong: *“even if something is wrong, the agents are very good at just going through—it will basically keep doing it, even though nothing went right.”*

A third limitation concerned the **limited scalability of LLM solutions across diverse use cases.** P3 reflected that current implementations are often tailored too narrowly: *“there is a big tendency to tailor a solution for one particular problem. But when we try to scale—not just in terms of load but in terms of variety of use cases—it starts to break. Fixing one issue often introduces unpredictable problems in new use cases.”*

C5. Organizational and Operational Constraints. Finally, participants pointed to challenges rooted in organizational context and data access. A key issue was the **limited availability of customer data.** As P1 explained, *“Finding relevant data is also a challenge because customer data we don’t have—the ones we can have are from third-party tools, which require approval and payment to use.”*

Participants highlighted **hosting concerns (e.g., data privacy and security).** Sensitive enterprise data could not always be moved into experimental environments. Relatedly, **enterprise restrictions in adopting commercial tools (e.g., legal approval due to user data privacy)** further limited experimentation with observability tools currently available on the market.

Table 2

Mapping between challenge subthemes, design goals, and XAgent's support. Empty cells mark desiderata scoped out of the prototype.

Challenge subtheme	Desired feature	Support in XAgent
C1.1 Steep learning curve and onboarding	G1.1 Clean log visualization; G4.2 log querying and summarization	✓ Log visualization
C1.2 Insufficient logging information	—	
C1.3 Managing large numbers of agents	—	
C2.1 Non-interruptible pipeline; tracking individual agent contributions	G2.1 run-from-breakpoint; G2.2 agent- and step-level contributions; G2.3 distributed unit testing	✓ Automatic error identification
C2.2 Issues from the broader code pipeline	G3.1 performance, resource, and tool-error tracking	—
C3.1 Tracking performance across the workflow over time	G3.1 performance, resource, and tool-error tracking	✓ Automatic error identification
C3.2 Unreliable metrics; reliance on domain expertise	G3.3 output check	✓ Human-in-the-loop output feedback
C4.1 Non-deterministic execution flow	G3.2 batch behavior profiling; G4.1 architecture-level recommendations	—
C4.2 Hallucination or incorrect outputs	G3.3 output check	✓ Human-in-the-loop output feedback with the LLM judge
C4.3 Limited scalability across use cases	G3.2 batch behavior profiling	—
C5.1–C5.3 Organizational constraints (data access, hosting, tool approval)	—	—

3.2.2. Desired Features for Explainability Tools

In response to the challenges participants faced when debugging their multi-agent systems, our formative interviews surfaced a set of desired design goals that practitioners considered important for developing explainability tools to support multi-agent systems. Table 2 summarizes how the design goals map onto the challenge themes.

G1. Interface improvements for easier onboarding and visibility. A recurring design goal was the need for a **clean visualization of logs**, including clear indications of the agent name, task, inputs/outputs, and communications. Such representations were seen as useful across different user groups but particularly valuable for AI novices. P5 emphasized that “*the special agent that I created and the role and so on... having that sort of schema set up somewhere, like ‘oh, this agent is asking and this agent is answering,’ really helps.*” P2 similarly noted the value of seeing the overall flow: “*If you have five agents, say like a manager coordinating a coding task in a software company, you can see where the process goes and how iterations happen... if you click on each of the nodes you can expand the input and output, and that would be really helpful.*”

G2. Agent- and step-level failure attribution. Participants stressed the importance of being able to pause and isolate parts of a workflow—features that offer users a clearer mental model of where errors emerge and would benefit not only experienced developers but also AI novices. One highly requested feature was **run-from-breakpoint** functionality, which would allow users to stop execution at a chosen step and inspect intermediate inputs and outputs. P6 explained, “*It would be really cool to have breakpoints. I could stop at a point and see the input and output so far in the pipeline. That way I’d know everything is working up to here, and then if I add a new agent I can see whether things break after that.*”

Participants also emphasized the need to evaluate **agent- and step-level contributions**, so that both novices and experts could better understand which agents, at which steps of the workflow, were responsible for successes or failures. P7 noted that failures often trace back to a single agent, and being able to isolate its behavior would make troubleshooting more manageable. P3 added that: “*Maybe if we have five agents interacting, only two actually made the change and the rest were just checking around... it’s interesting to measure the correlation between individual agent performance and group performance.*”

Finally, participants envisioned **standardized protocols for distributed unit testing** that would allow agents to be run, modified, and tested independently. P11 envisioned future systems where *“you could isolate a component, trigger individual agents from your own interface, and test them independently—almost like distributed unit testing.”*

G3. Output check. Participants emphasized the importance of systematic evaluation and tracking mechanisms to better understand agent and workflow performance. One desired capability was **agent and tool performance tracking with resource and time usage**. P7 highlighted the need to capture not only whether a tool was called successfully but also why errors occurred, such as calling the right tool with the wrong arguments.

A second request was **batch-level behavior tracking and profiling**. P7 explained, *“traditional debugging is done with a single input. Here we need to look at batches... an agent might look fine on one input, but across 10 or 20 examples it only works on a few.”* P2 envisioned tools that could track how agents performed on tasks over time: *“We need to somehow track—was it better or not, what worked and what didn’t, and learn from that history.”* Similarly, P3 suggested versioning support to capture changes in prompts or agent templates and link them to outcome differences.

Finally, participants stressed the need for **output checks**—both immediate detection of invalid inputs/outputs and downstream/intermediate performance metrics. Such checks were seen as helpful for AI novices as well as more experienced users. P3 added that correlating activity metrics (latency, steps, tool usage) with real-world outcomes could provide deeper insights into agent efficiency.

G4. Automated code optimization and log analysis. Participants also envisioned explainability tools that could actively assist with optimization and log analysis. One desired feature was **architecture-level optimization recommendations**, such as guidance on runtime, resource usage, and agent design. P1 explained that a debugging tool should provide actionable summaries, *“saying you spent too much money or time on this task, and suggesting how to optimize the prompt or merge multiple tool calls into one agent.”*

Another recurring idea was **querying and summarizing logs** to reduce the burden of manual inspection. P10 described wanting a conversational interface: *“I could ask, ‘find me logs related to execution for this URL yesterday’, and instead of manually searching, the system would summarize and tell me the story.”* They emphasized the potential of semantic search and vectorized representations of logs for troubleshooting. P8 suggested that: *“I can just say, ‘analyze this log, I suspect the issue is in this part of the code’, and the system could quickly scan the logs, narrow down the scope, and even suggest fixes.”* These accounts highlight a strong desire for LLM-powered support that can transform raw, text-heavy logs into actionable debugging guidance.

4. XAgent: Explainable Agentic AI Tool

Our formative study identified a broad set of desired goals for improving the explainability of multi-agent systems. In XAgent, we implemented three representative features that address both developer-oriented and domain-expert-oriented needs: (1) log visualization, (2) human-in-the-loop output feedback, and (3) automatic error identification. We prioritized these three features for two reasons. First, they directly target the challenge subthemes practitioners raised most frequently across both technical and non-technical user groups—onboarding and log interpretation (C1), attributing failures within non-interruptible pipelines (C2.1), and checking output quality without reliable metrics (C3.2)—while also detecting the downstream consequences of LLM-specific limitations such as hallucination (C4.2). Table 2 shows the full mapping. Second, they together form a closed loop for failure handling: log visualization supports detecting where a workflow fails (G1), automatic error identification localizes failures to specific tasks and agents (G2, G3), and human-in-the-loop feedback captures expert judgment that both corrects individual outputs and refines subsequent automatic evaluations. In contrast, G4 calls for automation capabilities that go beyond the explainability focus of this work and we thus scoped it to future work.

The prototype demonstrates how explainable multi-agent systems can support users with diverse levels of AI expertise. Figure 2 demonstrates a walkthrough of the interface. The technical architecture of our interface is illustrated in Figure 3, which will be explained in detail later in this section.

4.1. System Walkthrough

To make the interaction concrete, we walk through a debugging session in which a user improves a web-design workflow whose output renders as plain, unstyled HTML.

The user starts by selecting the project folder of the workflow (❶). Our prototype supports CrewAI projects, as practitioners in our formative study primarily relied on CrewAI in their daily work. XAgent parses the configuration files and renders the workflow as a flowchart of agents, tasks, and tools in the upper section of the central panel. Clicking

XAgent: Designing Explainability Support for Failure Detection and Correction in Multi-Agent Workflows

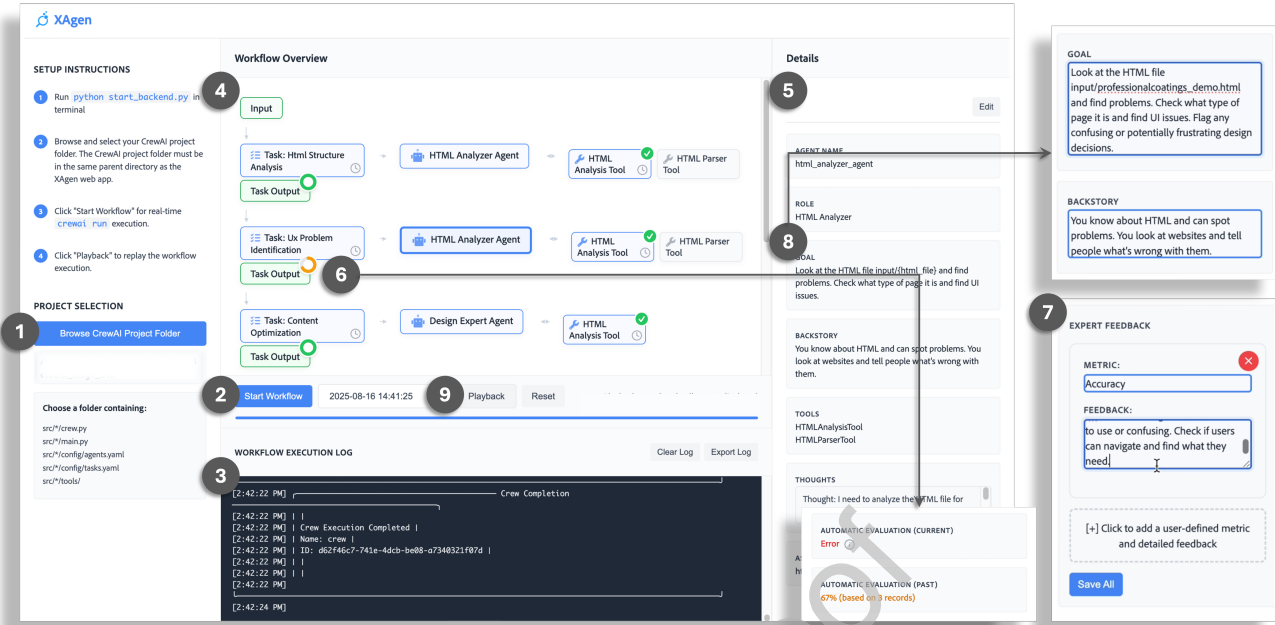


Figure 2: Walkthrough of the XAgent interface. The circled numbers mark the interaction steps: project selection and workflow execution (1–3), step-wise visualization and inspection (4–5), automatic evaluation (6), expert feedback and in-place configuration editing (7–8), and session replay (9).

Start Workflow (2) executes the multi-agent workflow: raw terminal logs stream into the lower section of the central panel (3), and each component in the flowchart is activated step by step in accordance with the logs (4), so the user can follow which agent is working on which task at any moment. Clicking a node opens the right panel with detailed information for that step, including prompt configurations, tool calls, and the agent's rationale (5).

When a task completes, its output is automatically evaluated by the LLM-as-a-judge (6). The average historical evaluation score is shown as a ring on the node, while detailed scores and rationales are listed in the right panel. In our scenario, the final web-design task is flagged. The judge's rationale states that the generated HTML links a stylesheet that is not present in the project, so the page renders without styling.

Instead of scanning the raw logs to locate this error, the user reads the rationale, confirms the problem in the task output, and records feedback in the manual feedback fields (7)—for example, that the deliverable must be a single self-contained page with embedded CSS. This feedback is stored and incorporated into subsequent judge evaluations. The user then edits the task's expected output directly in the interface (8) to require embedded styling and re-runs the workflow to test the improvement. In the new run, the task is evaluated positively and the styled page renders as intended. Finally, each session can be replayed (9), allowing the user to review how the workflow, evaluations, and feedback evolved across iterations.

4.2. Core Features and Implementation

4.2.1. Log Visualization

The log visualization feature was designed to directly address the usability challenges practitioners faced when onboarding and interpreting raw logs (C1). Rather than overwhelming users with verbose terminal outputs, the visualization represents each agent, task, input/output, and communication as clean, structured blocks, making the workflow easier to follow at a glance and serving as a foundation for locating errors at specific steps in the pipeline. This design choice was guided by practitioners' desiderata for interfaces that enable both novices and experienced users to quickly gain an overview of system behavior (G1). Figure 3b provides an overview of the XAgent log visualization component.

Configuration-Based Workflow Construction The system reads CrewAI configuration files to construct dynamic workflow flowcharts. Input sources include:

XAgent: Designing Explainability Support for Failure Detection and Correction in Multi-Agent Workflows

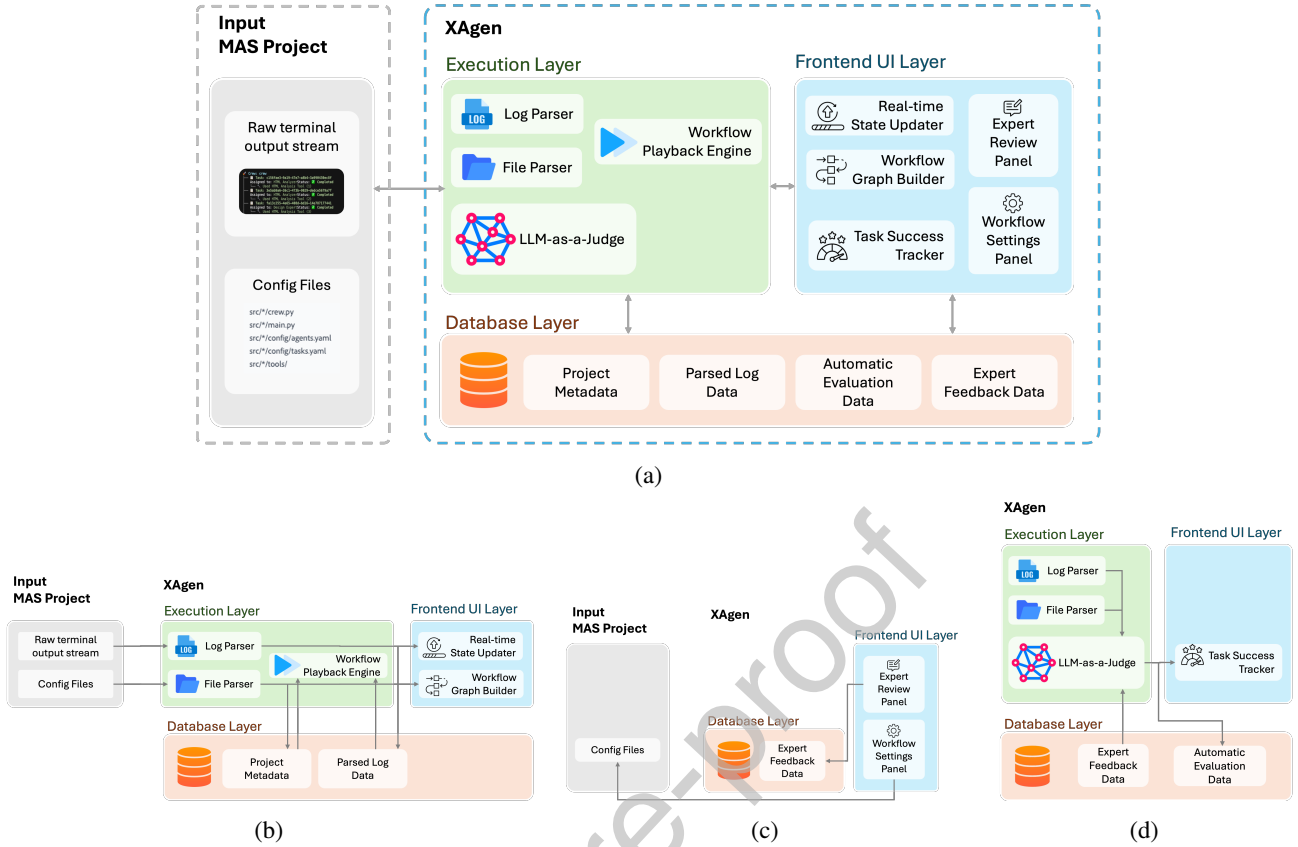


Figure 3: Architecture of XAgent (a), with bottom diagrams showing the detailed architecture of each core feature: log visualization (b), expert feedback (c), and automatic error detection (d). The XAgent operates in parallel with any multi-agent project and consists of three layers. The Database Layer stores essential data required for restoration of past runs. The Execution Layer provides backend functionality such as parsing logs and configuration files, generating workflow visualizations, and using the LLM-as-a-Judge for automatic evaluation. The Frontend UI Layer presents this information to users. Edits made in the UI are transferred back to the execution layer for processing.

- `src/*/crew.py` — Main workflow orchestration
- `src/*/main.py` — Entry point
- `src/*/config/agents.yaml` — Agent definitions
- `src/*/config/tasks.yaml` — Task definitions
- `src/*/tools/` — Available tool implementations

Real-Time Log Parsing The system implements a multi-threaded log parsing architecture that processes CrewAI execution streams in real time. The core parser uses regex pattern matching to extract structured data from unstructured terminal output. It monitors the `crewai run` subprocess output via `stdout` and `stderr` pipes and extracts entities such as task transitions, agent activations, and tool usage by identifying and categorizing log entries with specific regex patterns. For example:

- Agent activation: `r'# Agent:\s*(.+)'`
- Task execution: `r'## Task:\s*(.+)'`
- Tool usage: `r'Using tool:\s*(.+)'`
- Output generation: `r'## Final Answer:\s*(.+)'`

Frontend Synchronization Structured data and raw log entries are stored in JSON objects, then pushed to the frontend via WebSocket connections for real-time UI updates. The frontend visualization engine maintains workflow execution state with status indicators (complete/active/pending). Log events are mapped to visual elements (task/agent/tool/output nodes and status rings). Flowchart UI elements are dynamically activated as new log events arrive via WebSocket. This functionality is illustrated in Figure 2 as step 4.

Historical Data Storage and Playback The system implements a database to store raw logs, extracted structures, and execution sessions. The playback engine reconstructs historical workflow states by replaying stored events, allowing users to explore past executions, as shown in step 9 of Figure 2.

4.2.2. Human-in-the-Loop Output Feedback

The human-in-the-loop output feedback component was designed to allow users to directly validate and provide feedback on intermediate results. The feature addresses both the challenge of interpreting output quality in the absence of reliable metrics (C3) and the risks posed by hallucinated or incorrect model outputs (C4). It aligns with practitioners' desiderata for mechanisms that provide validation and quality checks (G3). Importantly, this human feedback is incorporated into the LLM-as-a-Judge component to automatically surface potential issues. Figure 3c provides an overview of the XAgent human-in-the-loop output feedback component. This expert feedback storage and configuration file modification process is demonstrated in Figure 2, step 7 and 8, respectively.

Expert Feedback Storage The interface provides an interactive UI for domain experts to review outputs and provide feedback. Feedback is linked to specific workflow components and stored in the database.

Configuration File Modification The system enables users to modify YAML configuration files directly through the interface by editing corresponding fields. Version control is handled via the database, which maintains timestamped backups of configuration files before applying expert-recommended changes. This architecture enables seamless integration of automated evaluation and human expertise for multi-agent workflow refinement.

4.2.3. Automatic Error Identification

The Automatic Error Identification feature was introduced to help practitioners localize problems without re-running entire pipelines, thereby addressing the difficulty of tracing errors in long, non-interruptible workflows (C2). It also mitigates the risk of hallucinated or incorrect results by automatically detecting invalid or inconsistent outputs (C4). This design reflects practitioners' desiderata for agent-level failure attribution (G2) as well as output checking (G3). In practice, the feature reduces debugging overhead by surfacing potential issues early and highlighting where further inspection or corrections are needed. Figure 3d provides an overview of the XAgent automatic error identification component.

The system includes an asynchronous evaluation pipeline that operates independently of the main workflow execution. When a task is completed and an output is generated, this triggers the evaluation pipeline. The system uses an LLM-as-a-Judge approach to compare task outputs against predefined goals and human feedback. XAgent then shows the flagged node with the judge's rationale, and users can trace backward along the flowchart, inspecting the inputs and outputs of preceding steps in the details panel to find where the omission was introduced.

The evaluation prompt template used in our study is provided in Appendix A. We set the temperature to 0 to ensure more deterministic outputs. A scoring algorithm with uncertainty handling was applied: "yes" were scored as 1.0 (task meets requirements), "no" as 0.0 (task fails requirements), and "unsure" as 0.5 (ambiguous evaluation). In addition to these scores, the LLM judge also generated a rationale for each evaluation to support interpretability. Users can also click on the information icon to access the detailed rationale provided by the LLM judge.

To help users monitor performance over time, the system computes moving averages across workflow executions for historical tracking. Visual indicators are displayed on the flowchart interface to reflect the aggregated success rates. See step 6 in Figure 2 for an illustration.

4.3. Technical Evaluation

We validated the LLM-as-a-Judge's judgments against author-verified ground-truth labels, as its effectiveness is central to the reliability of automatic error identification.

4.3.1. Setup

We constructed four multi-agent scenarios with CrewAI that mirror realistic workplace workflows: redesigning a poorly structured product landing page, writing a feature-launch marketing email, assembling a one-week social media content calendar, and analyzing customer reviews into an insights report. Each scenario comprises a full sequential workflow with 2–3 agents and 4–5 task steps. To generate final task outputs of varying quality, we prepared three instruction variants for each workflow: *correct* instructions that are clear and complete, *vague* instructions that omit format and content requirements, and *incorrect* instructions that embed factual errors, impossible constraints, or references to nonexistent resources. Each variant was executed 5 times with GPT-4o as the agent backbone, yielding 60 final task outputs.

For every output, the LLM judge received the *correct* task description and expected output together with a fixed expert feedback statement per scenario, and returned a yes/no/unsure label with a rationale using the prompt in Appendix A. The authors verified each of the 60 outputs against the correct specification to establish the ground truth independently of the instruction variants. Each output was checked against the requirements stated in the correct task description, aided by scripted checks for properties that can be verified mechanically, such as self-containment of the generated HTML, word and character limits, calendar-date validity, and whether quoted text appeared in the source reviews.

4.3.2. Results

The judge achieved 95.0% accuracy (57/60) against the verified labels. For flagging unacceptable outputs, precision was 97.7% (43/44) and recall was 95.6% (43/45). Given that 45 of the 60 outputs were unacceptable by construction, we note that the judge also correctly passed 14 of the 15 acceptable outputs, which a trivial always-reject baseline would misclassify. The judge correctly identified subtle failures, including fabricated statistics attributed to a nonexistent data file, marketing claims contradicting the product facts, and reports whose quotes did not appear verbatim in the source reviews. The three errors comprised two false acceptances (an email exceeding the word limit; a webpage referencing external image files) and one false rejection, in which the judge rejected a correct calendar with confused date reasoning. These results suggest that the LLM-as-a-Judge provides a reliable first-pass signal for failure detection, while its occasional errors reinforce the need for the human oversight that XAgent’s human-in-the-loop feedback supports.

5. User Evaluation

We conducted a usability evaluation to assess how users perceive XAgent’s support for identifying and correcting failures in multi-agent workflows. Specifically, we compared XAgent to a baseline debugging method in terms of perceived understanding, usability, helpfulness, and satisfaction. Our goal was to examine how participants with different levels of AI expertise experienced using XAgent to locate failures, attribute them to specific agents or steps, and iteratively improve workflow configurations.

Our evaluation was guided by three questions:

- **RQ1:** How do users perceive XAgent in terms of understanding, usability, helpfulness, and satisfaction when debugging multi-agent workflows?
- **RQ2:** How helpful do users perceive each of XAgent’s three features to be?
- **RQ3:** How do users with different levels of AI development experience use and experience these features during failure detection and correction?

5.1. Study Design

5.1.1. Participants

We recruited 8 participants from our organization, consisting of 3 female and 5 male participants (mean age = 28, SD = 1.6). The group included 7 graduate students from diverse majors and 1 software developer. Participants reported a wide range of prior experience with developing AI systems on a 5-point Likert scale (1 = none, 5 = extensive): 4 rated themselves as “extensive”, 1 as “none”, 2 as “a little”, and 1 as “some”. In other words, half of the participants had extensive AI development experience, while the other half—graduate students from majors outside computer science—had little to none, which allowed us to observe how users across this range experienced both systems.

5.1.2. Tasks

We designed two tasks to simulate realistic multi-agent workflows:

- **UI design crew.** In this task, participants worked with a workflow that attempted to improve a poorly designed webpage. The initial configuration was intentionally simple and produced outputs with limited appeal (for example, sometimes rendering as plain HTML due to missing CSS). Participants were asked to revise the configuration files (`tasks.yaml` and `agents.yaml`) by refining task descriptions, expected outputs, agent goals, and backstories. Their goal was to guide the agents toward generating webpages that were more visually appealing and engaging for end users.
- **Academic writing crew.** In this task, participants worked with a workflow that generated a related work section and a summary table given a research topic. The baseline configuration produced outputs that were overly simplistic (e.g., tables that only listed authors, years, and links). Participants were asked to edit the configuration files to encourage the agents to produce more structured, comprehensive, and organized related work sections, including more meaningful synthesis of prior studies.

5.1.3. Conditions

We compared XAgent with a baseline condition that combined raw system logs and AgentOps [1], a state-of-the-art observability tool. AgentOps provides visualization of logs and tracks technical details such as runtime and token usage, but it is primarily designed for developers and is less accessible to users with diverse levels of AI expertise.

5.1.4. Procedure

The study was conducted in person. At the beginning of each session, the experimenter introduced the study scenario and collected demographic information. Each participant completed both tasks using both systems (XAgent and the baseline), following a within-subjects design with counterbalanced order of systems and tasks.

Before starting, participants received a short guided tutorial for each system. For XAgent, the experimenter introduced its three core features (log visualization, automatic error identification, and human-in-the-loop feedback). For the Log+AgentOps baseline, participants were shown how to interpret terminal logs and the web-based AgentOps visualization.

During XAgent sessions, the experimenter also acted as a “domain expert” and provided example feedback through the expert feedback textbox (e.g., pointing out that a webpage output hallucinated a missing CSS file, or that a related work table should not only list author, year, and link). Participants then edited task and agent configurations directly in the XAgent interface. In contrast, during Log+AgentOps sessions, participants were instructed to edit only the configuration files within the project. Each session lasted around 20 minutes.

At the end of the two sessions, participants completed a post-study survey and answered a short set of open-ended interview questions. The sessions were recorded and transcribed to capture participant feedback. The entire evaluation session took approximately one hour.

5.2. Evaluation Methods

At the end of the study, participants completed a survey and a short follow-up interview to capture their experiences with both systems. We collected both quantitative ratings and qualitative feedback to evaluate XAgent.

Participants rated their experience on a set of 7-point Likert scales (1 = Log+AgentOps, 7 = XAgent, 4 = both equally) covering four key dimensions:

- **User understanding** [40, 49]: Which tool better helped you understand how each part of the agentic workflow led to a particular output?
- **Usability** [50]: Which tool did you find easier to learn and use?
- **Helpfulness** [11]: Which tool did you find more helpful for exploring and debugging agentic workflows?
- **User Satisfaction** [42, 50]: Which tool were you more satisfied with in terms of overall experience?

In addition, participants were asked to evaluate the usefulness of XAgent’s three individual features on separate 5-point Likert scales (1 = very unhelpful, 5 = very helpful) [11, 50].

After completing the tasks, participants were asked the following three open-ended questions in the interview:

XAgen: Designing Explainability Support for Failure Detection and Correction in Multi-Agent Workflows

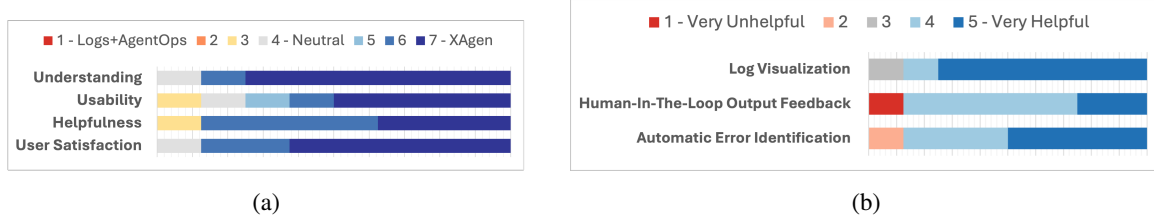


Figure 4: User evaluation results on four subjective ratings (a) and on helpfulness of XAgen’s three core features (b).

- What feature of XAgen did you find most helpful or enjoyable?
- Was there anything about the interface that felt overwhelming or confusing?
- Is there a part of the agentic workflow that you still do not understand or would like the interface to reveal you?

5.3. Results

5.3.1. Quantitative Findings

When compared with debugging using terminal logs and AgentOps, participants generally rated XAgen more positively across all four subjective metrics. Specifically, they reported that XAgen better supported their understanding of agentic workflows (median = 7.0, IQR = [6.75, 7.0]; one-sample Wilcoxon signed-rank test against 4 showed $p < 0.01$), was easier to learn and use (median = 6.5, IQR = [4.75, 7.0], $p < 0.05$), was more helpful for exploring and debugging workflows (median = 7.0, IQR = [6.0, 7.0], $p < 0.01$), and provided a more satisfying overall experience (median = 6.0, IQR = [6.0, 7.0], $p < 0.01$). These results (Figure 4a) suggest that participants perceived XAgen as better supporting them in locating failures, attributing them to specific agents or steps, and iteratively improving configurations compared to the baseline.

We also asked participants to evaluate the helpfulness of XAgen’s three core features on 5-point Likert scales. Across all three, the median ratings exceeded 3 (Figure 4b). Log Visualization received a median rating of 5/5 (IQR = [4.75, 5.0], $p < 0.01$) and Automatic Error Identification was rated 4.5/5 (IQR = [4.0, 5.0], $p < 0.05$), both significantly above the neutral point. Human-in-the-Loop Output Feedback was also rated above neutral (median = 4/5, IQR = [4.0, 4.25]), but only reached marginal significance ($p = 0.07$).

We further broke the ratings down by the two experience groups: 4 participants with extensive AI development experience and 4 with little to none. The two groups perceived XAgen similarly on most measures: understanding (median = 7.0 vs. 6.5), helpfulness (7.0 vs. 6.5), satisfaction (6.0 vs. 6.5), log visualization (5.0 in both groups), automatic error identification (4.5 in both), and human-in-the-loop feedback (4.0 vs. 4.5). The clearest difference concerned ease of learning: experienced participants uniformly favored XAgen over the baseline (median = 7.0), whereas ratings among less-experienced participants were mixed (median = 4.5, range 3–7). For users new to agent development, XAgen was thus not uniformly easier to pick up than the baseline—plausibly because the concepts underlying multi-agent workflows were themselves new.

5.3.2. Qualitative Findings

Log visualization. Log visualization enabled participants to gain a clearer understanding of the entire workflow during debugging. Participants explained that raw logs were often overly long, difficult to follow, and frequently truncated in the terminal, making it hard to trace problems. P6 summarized this frustration: “On the one hand, it’s hard for me to judge whether a log is good or not. On the other hand, logs aren’t really written for humans—they’re more like instructions for the next LLM task—so reading them as a human is very exhausting.”

XAgen’s structured visualization addressed these issues by displaying each agent, task, and tool as blocks in a workflow. This provided a holistic view that participants found far more intuitive than scanning through raw logs. P6, who didn’t have enough AI agent experience, explained that “This is exactly how I picture things in my head. But when I implement them myself, they usually end up as lines of code, and it’s hard to see the full picture. That’s a pain point for me... and now I can understand the overall architecture much more clearly.” Other participants echoed this benefit. P4 described how the feature helped them “quickly and easily see which agent performed what task at which step, along with the task results”, while also supporting on-the-fly configuration edits. P5 highlighted the clean interface and panel

separation, noting that it allowed them to “*immediately see which parts could be adjusted, view logs alongside other panels, and perform different operations with clear logic.*”

However, at present, both the logs and the workflow outputs are presented primarily in text. One recurring theme was the need to better visualize and simplify text-heavy outputs. Participants described feeling overwhelmed by the length of these logs and outputs, noting that large blocks of text discouraged them from carefully going through line by line. For example, P8 complained that there was “*a lot of texts*”, while P6 admitted that “*reading logs is really tiring... I can understand why there is so much text, but I cannot immediately think of a good way to make it better.*” Participants suggested that more intuitive formats could help reduce this burden. P3 pointed out that “*the final answers are too long—it would be helpful if they could be visualized*”, and P5 suggested that “*if intermediate outputs could be presented in a more intuitive way, it would really help.*” P1 hoped for a stronger “linking/jumping” feature, such that clicking a node would directly navigate to the corresponding log or file path.

Human-in-the-loop feedback. The human-in-the-loop feedback feature received relatively limited engagement in our study. This limitation was partly due to the study setting, where participants worked individually rather than collaboratively. As P5 explained, “*since I had never modified this part of the workflow and was unfamiliar with the related process, I wasn’t sure where to start. As a single user, if something went wrong, I didn’t always know how to fix it.*”

At the same time, participants recognized the value of this feature for supporting collaboration by enabling experts to review and intervene in workflow outputs. P1 described one of their pain points as, “*I want to optimize underperforming agents, but I don’t have a benchmark.*” They emphasized the usefulness of XAgent’s mechanism for incorporating expert judgment alongside automated processes, noting that if XAgent “*had a server to host it, then experts could provide feedback and conduct reviews very conveniently, which would make it well-suited for collaborative work.*” These reflections highlight the potential of XAgent not only as a debugging tool, but also as a platform for shared improvement in multi-agent workflows.

Automatic error identification. Automatic error identification helped participants locate errors and suggest configuration improvements more efficiently. This feature allowed users to directly determine whether a task was successfully completed. As agent developers, participants emphasized the importance of precisely tracing errors. P1 described a pain point: “*It takes a long time to run... I can’t identify the issue and have to re-run the whole thing.*”

The color-coded visualization of the LLM judge’s scores provided immediate feedback. As one participant put it, “*When I see it turn yellow or green, I instantly get an intuitive sense of what’s happening*”. P2, P5, and P6 all noted that the visualization helped them pinpoint which agent caused an error and quickly directed them to potential problems. Interestingly, some participants also preferred immediate feedback over cumulative ones. P2 questioned the value of historical evaluations: “*I think the past automatic evaluation is unnecessary. I don’t care about the past—I only care about the present. The past performance doesn’t matter anymore, since I’ve already iterated on it.*”

In addition to supporting error localization, the LLM judge also returned rationales for its evaluations. Participants sometimes incorporated text from these rationales into their prompt edits to improve workflow performance. P4 explained: “*The system provides an LLM judge to give an initial evaluation of the results. I think the judge’s feedback is often quite reasonable—sometimes even more thorough than my own.*”

6. Discussion

6.1. From Design Goals to Design Guidelines

The design goals in Section 3.2.2 (G1–G4) were derived from the formative interviews before XAgent was built. Here, we revisit those goals in light of the system implementation and evaluation, and draw out four guidelines that may be useful beyond XAgent. The guidelines focus on the goals that were both instantiated in the system and examined in our evaluation; goals that were not implemented are discussed separately as future directions.

DG1. Use the workflow structure as the main entry point for understanding execution. G1 asked for interfaces that make agent and task behavior easier to take in, concretely in the form of a clean visualization of logs showing agent names, tasks, inputs, and outputs (G1.1). Our implementation and evaluation suggest a stronger formulation of this goal: rather than requiring users to reconstruct the workflow from a trace, an explainability tool can make the workflow itself—its agents, tasks, tools, and dependencies—the main object of inspection, while retaining raw logs for details when needed. XAgent follows this approach by showing a persistent flowchart whose nodes are activated

as the workflow executes. Log visualization was the highest-rated feature in our evaluation, and participants with less agent-development experience described the structural view as consistent with how they already thought about the workflow. At the same time, locating a step did not necessarily make its contents easy to understand. Several participants still found the text shown within individual steps difficult to inspect. This suggests that workflow-level visualization and step-level content presentation are separate design problems: improving the former does not remove the need to simplify the latter.

DG2. Place automatic quality signals directly on the workflow. G2 asked for attributing failures to individual agents and steps, and G3 for checking whether task outputs meet their intended goals (G3.3). The two goals were elicited separately, but our implementation suggests that much of their value comes from combining them: placing the result of an output check on the workflow structure itself, rather than in a separate evaluation view, turns a quality signal into a pointer to where to look. In XAgent, the LLM judge evaluates each task output and displays the result on the corresponding workflow node, with a rationale available in the details panel. Participants rated automatic error identification highly, and several reported using the node indicators to decide where to inspect the workflow. The technical evaluation in Section 4.3 provides additional evidence that these per-task judgments can serve as a useful first-pass signal. However, they should not be interpreted as root-cause attribution. Because each task output is evaluated against that task's own specification, the system identifies where a problem becomes visible, but the underlying cause may lie in an earlier step. Per-step evaluation should therefore be paired with support for tracing upstream dependencies rather than presented as identifying the source of a failure.

DG3. Carry domain-expert feedback into subsequent evaluations. G3 also covered output checks in settings where no reliable metric exists and the judgment depends on domain knowledge that developers or system operators may not have (G3.3). This suggests that human feedback should not terminate as a standalone annotation. XAgent therefore stores expert feedback and includes it in the LLM judge's context on later runs, so that feedback can influence subsequent automatic evaluations. The user study, however, provided limited evidence for the usefulness of this mechanism. Human-in-the-loop output feedback received a median rating of 4/5 but only reached marginal significance ($p = 0.07$), and participants used it less than the other two features. Their comments suggest an important reason: participants completed the study individually, while the feature is most relevant when the person operating the system and the person providing domain expertise are not the same. Participants nevertheless saw potential value in allowing domain experts to review and comment on outputs. We therefore view persistent expert feedback primarily as a collaborative mechanism, whose value should be evaluated in settings where domain experts and system builders work together.

DG4. Support inspection and revision in the same workspace. G1–G3 were elicited as separate capabilities, but practitioners described them as parts of a single loop. G2.1, for instance, asked for the ability to pause at a chosen step and continue from there, reflecting a cost that extended beyond finding a problem: fixing it often required leaving the observability tool, editing configuration files elsewhere, re-running the workflow, and then returning to the logs to inspect the result. XAgent reduces these context switches by allowing users to inspect outputs, edit agent and task configurations, re-run the workflow, and review previous runs within the same interface. Participants made configuration changes while inspecting the workflow and rated the overall usability and satisfaction of the system highly. However, the interaction remained coarse-grained: testing a change still required re-running the full workflow. Participants repeatedly pointed to this cost, suggesting that tighter integration of inspection and editing should ultimately be paired with mechanisms such as restarting from an intermediate step.

Not all of the goals identified in the formative study are represented in these guidelines. We did not implement run-from-breakpoint or distributed unit testing (G2.1, G2.3), batch behavior profiling (G3.2), or automated code optimization and log analysis (G4), and therefore do not have evaluation evidence for them. We retain these findings as directions raised by practitioners rather than as validated design guidelines or capabilities of XAgent.

6.2. System Improvements

Our system can benefit from design extensions informed by user evaluation feedback. A recurring need was to better visualize and simplify text-heavy intermediate and final outputs from agentic workflows. More intuitive formats could help reduce this burden. First, intermediate results could be presented more concisely, for example by extracting key entities from text [43, 18, 49]. Another direction is to incorporate visual indicators directly into the interface [37]. For instance, bounding boxes on a webpage could highlight problematic areas and allow errors to be located more quickly [17], and descriptive tours of key snippets could provide a coherent narrative tracing of an AI agent's exploration path [46]. Finally, in design-oriented tasks, having preview images generated at intermediate steps would be useful [24],

as it would allow users to confirm whether a UI layout was correct earlier in the process, catch problems sooner and reduce unnecessary rework.

Second, XAgent's human feedback module shows potential for supporting collaborative work. If deployed on a server-based shared platform, domain experts could conveniently review and intervene in workflow outputs. This collaborative potential warrants further investigation in multi-user contexts.

Finally, although the LLM judge provided rationales for its evaluations, the system currently lacks features such as auto-fill suggestions for prompt revisions, which could further support more efficient iteration.

6.3. Limitations

There are several limitations of the current prototype. First, XAgent only supports sequential crew structures. Participants were able to edit agent and task configurations by modifying text in the side panel, but they could not change the overall structure—for example, adding or deleting tasks, creating hierarchical crews, reordering tasks or agents, or modifying tools. Enabling such functionality would make exploration more flexible. Future iterations should support non-linear workflows, such as loops, hierarchies, and parallel execution, to handle more complex real-world agentic collaborations. Second, our user evaluation did not take place in a collaborative environment. Each participant used the system individually, so they were unable to fully experience and evaluate the human-in-the-loop expert feedback feature. Moreover, because the experimenter demonstrated the expert-feedback feature during XAgent sessions, the baseline condition received no equivalent demonstration, which may have contributed to XAgent's favorable comparative ratings. In addition, our evaluation relied on subjective comparative ratings. Both study tasks were open-ended design and writing tasks without a verifiable ground truth, participants iterated on webpage aesthetics and literature-summary quality until satisfied, so behavioral measures such as error-localization accuracy were not well defined in this setting. Evaluations with tasks that afford objective ground truth remain future work. Relatedly, the system did not provide sufficient technical details, such as runtime, token usage, or monetary cost, which could be valuable for expert debuggers. While logs could include user-defined information, the prototype only supported parsing default CrewAI logs. Finally, although the LLM judge offered rationales for its evaluations, the system lacked features such as auto-fill suggestions for prompt revisions that could have further reduced user effort.

6.4. Future Work

Our study points to several open directions for future research and system development for explainable multi-agent systems. First, agents' decisions at each step often appeared quite stochastic. Stabilizing these decisions and providing consistent justifications would improve system reliability. For instance, the system should be able to offer background explanations for tool selection and surface alternative options rather than presenting only one stochastic result.

Second, there is a need to better isolate tasks and agent contributions. A recurring frustration was that the system currently requires rerunning the entire workflow, even when users only wish to test a single task. Model execution is often slow and resource-intensive. The capability to selectively re-run individual tasks or agents, followed by an automatic reevaluation of results, would save time and effort. Related to this, there would be longer waits during long executions. An open design question is therefore: what should users do during this waiting period? Should they watch the execution progress step by step, or should the system support them in switching to other relevant activities while keeping them informed of progress? This is an underexplored design space of making the waiting experience more productive.

Finally, future work should further strengthen the evaluation of XAgent. Using an ablated version of XAgent—with features systematically removed one at a time—as a baseline would better isolate each feature's contribution; and incorporating objective measures such as time to identify errors and correction accuracy would make the evaluation more rigorous. Although participants spanned a wide range of AI development experience, non-technical domain experts who informed the formative study were not represented in the evaluation. Our findings therefore speak to users with varying levels of AI development experience rather than the full spectrum of non-technical users that motivates our design, whose evaluation we leave to future deployment studies.

7. Conclusion

As LLM-powered multi-agent systems are increasingly applied in both individual and enterprise settings, it becomes critical for users to not only build these systems but also identify and correct failures in their workflows. Yet, current debugging methods remain difficult to onboard for developers and provide little support for users with

limited AI expertise. In this work, we extend the concept of XAI to multi-agent systems to assist users in identifying and correcting failures. Grounded in formative interviews with real-world practitioners with diverse AI expertise, we designed XAgent, an explainability tool that integrates log visualization for glanceable workflow understanding, human-in-the-loop feedback to capture expert judgment, and automatic error identification with an LLM as a judge. A technical evaluation of the LLM judge and a usability evaluation with 8 participants show that the judge reliably flags failing outputs and that users perceived the three features as supporting failure detection and correction, while also revealing open challenges. We envision this work as a step toward the human-centered design of explainability interfaces that enable even novice users to participate in building and refining agentic workflows in an era where AI increasingly collaborates with humans.

Acknowledgements

We would like to thank our colleagues at Adobe Research and the *Adobe LLM Optimizer* and *AEM Sites Optimizer* teams for their thoughtful feedback and support throughout this project. We are also grateful to all participants who contributed their time to our formative study and user study. Finally, we thank all anonymous reviewers for their insightful and constructive comments. This research has been partially supported by the National Research Foundation (NRF), Prime Minister's Office, Singapore, under its Campus for Research Excellence and Technological Enterprise (CREATE) program, through the Mens, Manus and Machina (M3S) interdisciplinary research group (IRG) of the Singapore-MIT Alliance for Research and Technology (SMART) center.

A. LLM-as-a-Judge Prompt

You are an impartial evaluator. Your task is to assess whether the FINAL ANSWER generated by an AI agent adequately satisfies the given TASK DESCRIPTION and EXPECTED OUTPUT, taking into account EXPERT FEEDBACK from previous attempts. Please return your judgment in the following JSON format:

```
{
  "label": "yes" | "no" | "unsure",
  "rationale": "<brief explanation of your reasoning>"
}
```

Inputs:

TASK DESCRIPTION: {task_description}

EXPECTED OUTPUT: {expected_output}

EXPERT FEEDBACK: {expert_feedback}

FINAL ANSWER: {final_answer}

Important: Respond with only the JSON object. Do not include any additional text or commentary.

CRedit authorship contribution statement

Xinru Wang: Conceptualization, Investigation, Data curation, Formal analysis, Methodology, Software, Visualization, Writing - Original Draft. **Zhuoran Lu:** Writing - Review & Editing. **Shuai Ma:** Writing - Review & Editing. **Ming Yin:** Writing - Review & Editing. **Eunye Koh:** Funding acquisition, Project administration. **Mustafa Doga Dogan:** Conceptualization, Investigation, Project administration, Supervision.

CRedit authorship contribution statement

Xinru Wang: Conceptualization, Investigation, Data curation, Formal analysis, Methodology, Software, Visualization, Writing - Original Draft. Zhuoran Lu: Writing - Review & Editing. Shuai Ma: Writing - Review & Editing. Ming Yin: Writing - Review & Editing. Eunye Koh: Funding acquisition, Project administration. Mustafa Doga Dogan: Conceptualization, Investigation, Project administration, Supervision.

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Journal Pre-proof

References

- [1] AgentOps.ai, 2025. AgentOps. URL: <https://www.agentops.ai/>.
- [2] Arawjo, I., Swoopes, C., Vaithilingam, P., Wattenberg, M., Glassman, E., 2024. ChainForge: A Visual Toolkit for Prompt Engineering and LLM Hypothesis Testing, in: Proceedings of the CHI Conference on Human Factors in Computing Systems, pp. 1–18. URL: <http://arxiv.org/abs/2309.09128>, doi:10.1145/3613904.3642016. arXiv:2309.09128 [cs].
- [3] CrewAI, 2025. crewAIInc/crewAI: Framework for orchestrating role-playing, autonomous AI agents. By fostering collaborative intelligence, CrewAI empowers agents to work together seamlessly, tackling complex tasks. URL: <https://github.com/crewAIInc/crewAI>.
- [4] Dhanoa, V., Wolter, A., León, G.M., Schulz, H.J., Elmqvist, N., 2025. Agentic Visualization: Extracting Agent-based Design Patterns from Visualization Systems. URL: <http://arxiv.org/abs/2505.19101>, doi:10.48550/arXiv.2505.19101. arXiv:2505.19101 [cs].
- [5] Dibia, V., Chen, J., Bansal, G., Syed, S., Fournery, A., Zhu, E., Wang, C., Amershi, S., 2024. AutoGen Studio: A No-Code Developer Tool for Building and Debugging Multi-Agent Systems. URL: <http://arxiv.org/abs/2408.15247>, doi:10.48550/arXiv.2408.15247. arXiv:2408.15247 [cs].
- [6] Dierk, C., Healey, J., Dogan, M.D., 2025. Evaluating LLMs in Experiential Context: Insights from a Survey of Recent CHI Publications .
- [7] Ding, S., Chen, X., Fang, Y., Liu, W., Qiu, Y., Chai, C., 2023. DesignGPT: Multi-Agent Collaboration in Design, in: 2023 16th International Symposium on Computational Intelligence and Design (ISCID), pp. 204–208. URL: <https://ieeexplore.ieee.org/document/10494260>, doi:10.1109/ISCID59865.2023.00056.
- [8] Ding, Z., Zhang, Q., Chi, M., Wang, Z., 2025. Frontend Diffusion: Empowering Self-Representation of Junior Researchers and Designers Through Multi-agent System. URL: <http://arxiv.org/abs/2502.03788>, doi:10.48550/arXiv.2502.03788. arXiv:2502.03788 [cs] version: 2.
- [9] Drago, R., Sechayk, Y., Dogan, M.D., Sanna, A., Igarashi, T., 2025. ImprovMate: Multimodal AI Assistant for Improv Actor Training, in: Companion Publication of the 2025 ACM Designing Interactive Systems Conference, ACM, Funchal Portugal. pp. 526–532. URL: <https://dl.acm.org/doi/10.1145/3715668.3736363>, doi:10.1145/3715668.3736363.
- [10] Ehsan, U., Passi, S., Liao, Q.V., Chan, L., Lee, I.H., Muller, M., Riedl, M.O., 2024. The Who in XAI: How AI Background Shapes Perceptions of AI Explanations, in: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems, Association for Computing Machinery, New York, NY, USA. pp. 1–32. URL: <https://dl.acm.org/doi/10.1145/3613904.3642474>, doi:10.1145/3613904.3642474.
- [11] Epperson, W., Bansal, G., Dibia, V.C., Fournery, A., Gerrits, J., Zhu, E.E., Amershi, S., 2025. Interactive Debugging and Steering of Multi-Agent AI Systems, in: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, Association for Computing Machinery, New York, NY, USA. pp. 1–15. URL: <https://dl.acm.org/doi/10.1145/3706598.3713581>, doi:10.1145/3706598.3713581.
- [12] Fournery, A., Bansal, G., Mozannar, H., Tan, C., Salinas, E., Erkan, Z., Niedtner, F., Proebsting, G., Bassman, G., Gerrits, J., Alber, J., Chang, P., Loynd, R., West, R., Dibia, V., Awadallah, A., Kamar, E., Hosn, R., Amershi, S., 2024. Magentic-One: A Generalist Multi-Agent System for Solving Complex Tasks. URL: <http://arxiv.org/abs/2411.04468>, doi:10.48550/arXiv.2411.04468. arXiv:2411.04468 [cs].
- [13] Google, 2025. Agent2Agent (A2A) Protocol Documentation. URL: <https://a2a-protocol.org/latest/>.
- [14] Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N.V., Wiest, O., Zhang, X., 2024. Large Language Model Based Multi-agents: A Survey of Progress and Challenges, pp. 8048–8057. URL: <https://www.ijcai.org/proceedings/2024/890>, doi:10.24963/ijcai.2024/890.
- [15] He, H., Yao, W., Ma, K., Yu, W., Dai, Y., Zhang, H., Lan, Z., Yu, D., 2024. WebVoyager: Building an End-to-End Web Agent with Large Multimodal Models, in: Ku, L.W., Martins, A., Srikumar, V. (Eds.), Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Bangkok, Thailand. pp. 6864–6890. URL: <https://aclanthology.org/2024.acl-long.371/>, doi:10.18653/v1/2024.acl-long.371.
- [16] Holter, S., Koh, E., Dogan, M.D., Chan, G.Y.Y., 2026. UXCascade: Scalable Usability Testing with Simulated User Agents. URL: <http://arxiv.org/abs/2601.15777>, doi:10.48550/arXiv.2601.15777. arXiv:2601.15777 [cs].
- [17] Hoque, M.N., Mashiat, T., Ghai, B., Shelton, C.D., Chevalier, F., Kraus, K., Elmqvist, N., 2024. The HaLLMark Effect: Supporting Provenance and Transparent Use of Large Language Models in Writing with Interactive Visualization, in: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems, Association for Computing Machinery, New York, NY, USA. pp. 1–15. URL: <https://dl.acm.org/doi/10.1145/3613904.3641895>, doi:10.1145/3613904.3641895.
- [18] Jiang, P., Rayan, J., Dow, S.P., Xia, H., 2023. Graphologue: Exploring Large Language Model Responses with Interactive Diagrams, in: Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, Association for Computing Machinery, New York, NY, USA. pp. 1–20. URL: <https://dl.acm.org/doi/10.1145/3586183.3606737>, doi:10.1145/3586183.3606737.
- [19] Jung, J., Concannon, C., Shroff, R., Goel, S., Goldstein, D.G., 2017. Simple rules for complex decisions. URL: <http://arxiv.org/abs/1702.04690>, doi:10.48550/arXiv.1702.04690. arXiv:1702.04690 [stat].
- [20] Kim, B., Khanna, R., Koyejo, O.O., 2016. Examples are not enough, learn to criticize! Criticism for Interpretability, in: Advances in Neural Information Processing Systems, Curran Associates, Inc. URL: https://papers.nips.cc/paper_files/paper/2016/hash/5680522b8e2bb01943234bce7bf84534-Abstract.html.
- [21] Lakkaraju, H., Bach, S.H., Leskovec, J., 2016. Interpretable Decision Sets: A Joint Framework for Description and Prediction, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, New York, NY, USA. pp. 1675–1684. URL: <https://doi.org/10.1145/2939672.2939874>, doi:10.1145/2939672.2939874.
- [22] LangChain, 2025. langchain-ai/langgraph. URL: <https://github.com/langchain-ai/langgraph>. original-date: 2023-08-09T18:33:12Z.
- [23] Langtrace, 2025. Langtrace. URL: <https://www.langtrace.ai/>.

XAgent: Designing Explainability Support for Failure Detection and Correction in Multi-Agent Workflows

- [24] Li, R., Zhang, Y., Yang, D., 2025. Sketch2Code: Evaluating Vision-Language Models for Interactive Web Design Prototyping, in: Chiruzzo, L., Ritter, A., Wang, L. (Eds.), Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), Association for Computational Linguistics, Albuquerque, New Mexico. pp. 3921–3955. URL: <https://aclanthology.org/2025.naacl-long.198/>, doi:10.18653/v1/2025.naacl-long.198.
- [25] Liu, Y., Chen, S., Cheng, H., Yu, M., Ran, X., Mo, A., Tang, Y., Huang, Y., 2024. How AI Processing Delays Foster Creativity: Exploring Research Question Co-Creation with an LLM-based Agent, in: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems, Association for Computing Machinery, New York, NY, USA. pp. 1–25. URL: <https://dl.acm.org/doi/10.1145/3613904.3642698>, doi:10.1145/3613904.3642698.
- [26] Lundberg, S.M., View Profile, Lee, S.I., View Profile, 2017. A unified approach to interpreting model predictions, in: Proceedings of the 31st International Conference on Neural Information Processing Systems. Guide Proceedings, pp. 4768–4777. URL: <https://dl.acm.org/doi/10.5555/3295222.3295230>, doi:10.5555/3295222.3295230.
- [27] Microsoft, 2025. microsoft/autogen. URL: <https://github.com/microsoft/autogen>. original-date: 2023-08-18T11:43:45Z.
- [28] OpenAI, 2024. GPT-4 Technical Report. URL: <http://arxiv.org/abs/2303.08774>, doi:10.48550/arXiv.2303.08774. arXiv:2303.08774 [cs].
- [29] Protocol, M.C., 2025. Introduction - Model Context Protocol. URL: <https://modelcontextprotocol.io/docs/getting-started/intro>.
- [30] Ribeiro, M.T., Singh, S., Guestrin, C., 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, New York, NY, USA. pp. 1135–1144. URL: <https://dl.acm.org/doi/10.1145/2939672.2939778>, doi:10.1145/2939672.2939778.
- [31] Rieder, T., Schneider, M., Truss, M., Tsaplin, V., Rublea, A., Dere, S., Sanz, F.C., Reiss, T., Dogan, M.D., 2026. SimAB: Simulating A/B Tests with Persona-Conditioned AI Agents for Rapid Design Evaluation. URL: <http://arxiv.org/abs/2603.01024>, doi:10.48550/arXiv.2603.01024. arXiv:2603.01024 [cs].
- [32] Suresh, H., Gomez, S.R., Nam, K.K., Satyanarayan, A., 2021. Beyond Expertise and Roles: A Framework to Characterize the Stakeholders of Interpretable Machine Learning and their Needs, in: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, Association for Computing Machinery, New York, NY, USA. pp. 1–16. URL: <https://dl.acm.org/doi/10.1145/3411764.3445088>, doi:10.1145/3411764.3445088.
- [33] Tatsukawa, Y., Shen, I.C., Dogan, M.D., Qi, A., Koyama, Y., Shamir, A., Igarashi, T., 2025. FontCraft: Multimodal Font Design Using Interactive Bayesian Optimization, in: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, ACM, Yokohama Japan. pp. 1–14. URL: <https://dl.acm.org/doi/10.1145/3706598.3713863>, doi:10.1145/3706598.3713863.
- [34] Technologies, F., 2025. Langfuse. URL: <https://langfuse.com/>.
- [35] Voita, E., Ferrando, J., Nalmpantis, C., 2024. Neurons in Large Language Models: Dead, N-gram, Positional, in: Ku, L.W., Martins, A., Srikumar, V. (Eds.), Findings of the Association for Computational Linguistics: ACL 2024, Association for Computational Linguistics, Bangkok, Thailand. pp. 1288–1301. URL: <https://aclanthology.org/2024.findings-acl.75/>, doi:10.18653/v1/2024.findings-acl.75.
- [36] Wachter, S., Mittelstadt, B., Russell, C., 2018. Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR. URL: <http://arxiv.org/abs/1711.00399>, doi:10.48550/arXiv.1711.00399. arXiv:1711.00399 [cs].
- [37] Wang, J., Lan, H., Su, X., Dogan, M.D., Wang, A.Y., 2026. UI Remix: Supporting UI Design Through Interactive Example Retrieval and Remixing. URL: <http://arxiv.org/abs/2601.18759>, doi:10.1145/3742413.3789154. arXiv:2601.18759 [cs].
- [38] Wang, X., Kim, H., Rahman, S., Mitra, K., Miao, Z., 2024. Human-LLM Collaborative Annotation Through Effective Verification of LLM Labels, in: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems, Association for Computing Machinery, New York, NY, USA. pp. 1–21. URL: <https://doi.org/10.1145/3613904.3641960>, doi:10.1145/3613904.3641960.
- [39] Wang, X., Liang, C., Yin, M., 2023. The Effects of AI Biases and Explanations on Human Decision Fairness: A Case Study of Bidding in Rental Housing Markets. pp. 3076–3084. URL: <https://www.ijcai.org/proceedings/2023/343>, doi:10.24963/ijcai.2023/343.
- [40] Wang, X., Yin, M., 2021. Are Explanations Helpful? A Comparative Study of the Effects of Explanations in AI-Assisted Decision-Making, in: Proceedings of the 26th International Conference on Intelligent User Interfaces, Association for Computing Machinery, New York, NY, USA. pp. 318–328. URL: <https://dl.acm.org/doi/10.1145/3397481.3450650>, doi:10.1145/3397481.3450650.
- [41] Wang, X., Yin, M., 2022. Effects of Explanations in AI-Assisted Decision Making: Principles and Comparisons. ACM Trans. Interact. Intell. Syst. 12, 27:1–27:36. URL: <https://dl.acm.org/doi/10.1145/3519266>, doi:10.1145/3519266.
- [42] Wang, X., Yin, M., 2023. Watch Out for Updates: Understanding the Effects of Model Explanation Updates in AI-Assisted Decision Making, in: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, Association for Computing Machinery, New York, NY, USA. pp. 1–19. URL: <https://dl.acm.org/doi/10.1145/3544548.3581366>, doi:10.1145/3544548.3581366.
- [43] Wang, X., Yu, M., Nguyen, H., Iuzzolino, M., Wang, T., Tang, P., Lynova, N., Tran, C., Zhang, T., Sendhilnathan, N., Benko, H., Xia, H., Jonker, T.R., 2025. Less or More: Towards Glanceable Explanations for LLM Recommendations Using Ultra-Small Devices, in: Proceedings of the 30th International Conference on Intelligent User Interfaces, ACM, Cagliari Italy. pp. 938–951. URL: <https://dl.acm.org/doi/10.1145/3708359.3712074>, doi:10.1145/3708359.3712074.
- [44] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E.H., Le, Q.V., Zhou, D., 2022. Chain-of-thought prompting elicits reasoning in large language models, in: Proceedings of the 36th International Conference on Neural Information Processing Systems, Curran Associates Inc., Red Hook, NY, USA. pp. 24824–24837.
- [45] Wooldridge, M., Jennings, N.R., 1995. Intelligent agents: theory and practice. The Knowledge Engineering Review 10, 115–152. URL: https://www.cambridge.org/core/product/identifier/S0269888900008122/type/journal_article, doi:10.1017/S0269888900008122.
- [46] Yan, L., Tao, J., Chilton, L., Head, A., 2025. Answering Developer Questions with Annotated Agent-Discovered Program Traces, in: The 38th Annual ACM Symposium on User Interface Software and Technology. doi:10.1145/3746059.3747652.

XAgent: Designing Explainability Support for Failure Detection and Correction in Multi-Agent Workflows

- [47] Yang, J., Jimenez, C.E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K.R., Press, O., 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. URL: [https://openreview.net/forum?id=mXpq6ut8J3&referrer=%5Bthe%20profile%20of%20Shunyu%20Yao%5D\(%2Fprofile%3Fid%3D%5C~Shunyu_Yao1\)](https://openreview.net/forum?id=mXpq6ut8J3&referrer=%5Bthe%20profile%20of%20Shunyu%20Yao%5D(%2Fprofile%3Fid%3D%5C~Shunyu_Yao1)).
- [48] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y., 2023. ReAct: Synergizing Reasoning and Acting in Language Models. URL: <http://arxiv.org/abs/2210.03629>, doi:10.48550/arXiv.2210.03629. arXiv:2210.03629 [cs].
- [49] Zhang, A., Bursztyn, V., Chan, G., Guo, S., Koh, E., Oney, S., Hoffswell, J., 2025. ConvoMap: Interactive Visualizations for Exploring Complex Conversations in Multi-Agent Systems, in: 2025 IEEE Symposium on Visual Languages and Human-Centric Computing.
- [50] Zhang, J., Arawjo, I., 2025. ChainBuddy: An AI Agent System for Generating LLM Pipelines, in: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, pp. 1–21. URL: <http://arxiv.org/abs/2409.13588>, doi:10.1145/3706598.3714085. arXiv:2409.13588 [cs].
- [51] Zhang, Z., Zhang, A., Li, M., Zhao, H., Karypis, G., Smola, A., 2024. Multimodal Chain-of-Thought Reasoning in Language Models. URL: <http://arxiv.org/abs/2302.00923>, doi:10.48550/arXiv.2302.00923. arXiv:2302.00923 [cs].
- [52] Zou, A., Phan, L., Chen, S., Campbell, J., Guo, P., Ren, R., Pan, A., Yin, X., Mazeika, M., Dombrowski, A.K., Goel, S., Li, N., Byun, M.J., Wang, Z., Mallen, A., Basart, S., Koyejo, S., Song, D., Fredrikson, M., Kolter, J.Z., Hendrycks, D., 2025. Representation Engineering: A Top-Down Approach to AI Transparency. URL: <http://arxiv.org/abs/2310.01405>, doi:10.48550/arXiv.2310.01405. arXiv:2310.01405 [cs].