ConvoMap: Interactive Visualizations for Exploring Complex Conversations in Multi-Agent Systems

Ashley Ge Zhang[1], Victor Bursztyn[2], Gromit Chan[2], Shunan Guo[2],
Eunyee Koh[2], Steve Oney[1], and Jane Hoffswell[2]
[1]*University of Michigan, Ann Arbor, USA;* [2]*Adobe Research, USA*
{gezh, soney}@umich.edu, {soaresbu, ychan, sguo, eunyee, jhoffs}@adobe.com

***Abstract*— Following the rapid emergence of large language models, Multi-Agent Systems (MASs) became a promising approach for accomplishing complex tasks. In MASs, multiple autonomous agents with predetermined roles collaborate by dividing responsibilities. However, MAS developers often struggle to understand and diagnose agents' behavior from thousands of inter-agent messages across multiple complex conversations. To identify key requirements and challenges related to evaluating, debugging, and managing MASs, we conducted a formative study with six MAS developers. We then introduce ConvoMap, a prototype that addresses a key challenge of MAS development—understanding agents' behaviors across multiple conversations. ConvoMap integrates automated qualitative coding to enable multi-level inspection of agents' behavior. ConvoMap can then visualize hundreds of MAS conversations by representing messages as points on a 2D map that encode their semantic meanings and interactions between agents. To better support navigation and deeper analysis, ConvoMap provides topic overviews and highlights relevant text segments. A comparison study showed that ConvoMap helped to understand agents' behavior more accurately than the baseline.**

***Index Terms*—AI agents, multi-agent systems, AI debugging, language models**

I. Introduction

Multi-Agent Systems (MASs) contain multiple autonomous agents that learn from the environment and collaborate to solve complex tasks [1]. Their collaborative nature makes them well-suited for fields such as robotics [2], [3], language learning [4], programming [5], [6], and human behavior simulation [7]. Recently, MASs have leveraged large language models (LLMs) to handle real-world tasks such as SQL synthesis [8] and text-to-video generation [9], enabling greater automation, flexibility [10], and decision-making in complex workflows [11].

Despite the benefits of multi-agent AI systems [5], [7], current tools offer limited support for managing agent conversations and evaluating behavior at scale. Unlike traditional programs, MAS interactions are unpredictable due to LLM inconsistencies [12], [13], and agents are often treated as black boxes, with behavior only observable through their message outputs. Existing methods for evaluating single-agent LLMs—such as prompt tuning and task-specific criteria [14]–[16]—do not address MAS-specific challenges, including error propagation between agents and flawed task coordination. MAS development also involves extensive testing and iteration, which generates large volumes of conversation logs in natural language and code, making it more time-consuming to analyze agents' behavior across runs. Moreover, MAS development often includes collaboration between developers (who understand the system's internals) and domain experts (who assess only the outputs), leading to mismatches in evaluation needs. While tools like AGDebugger support debugging individual MAS workflows by editing and testing hypotheses about agent behavior [17], they lack the ability to provide overviews of large-scale MAS behavior or comparisons across different development settings.

We interviewed six MAS developers to understand their current MAS development workflows, key challenges, and how they manage and interpret extensive agent conversations. Consistent with prior work, our findings confirm the difficulty of understanding long conversations that emerge as MAS executes tasks, as well as the lack of debugging support in existing tools. Extending these insights, our findings revealed additional challenges in understanding large volumes of conversation data across multiple runs and supporting evaluation needs spanning both final results and intermediate outputs. To address these challenges, systems must support comparison of LLM outputs at multiple levels, flexible message grouping, and intuitive visualizations of message semantics.

Based on our findings from these interviews, we introduce ConvoMap, a visual analytics tool that enables developers to explore large-scale MAS conversations using a 2D map. ConvoMap visualizes multi-agent conversations as flows, capturing both temporal and semantic dynamics as topics shift. Agent messages are represented as circles on the map that encode semantic similarity, task progress, division, and behavioral evolution. To digest the text-heavy content into the system, ConvoMap integrates automated qualitative coding and supports multi-level message grouping—from full conversations to individual segments—helping users understand agent behavior, diagnose issues, and compare interactions at scale.

We evaluated ConvoMap with 16 participants and over 200 MAS conversations from two MASs. Our findings show that ConvoMap enabled more accurate understanding of agent behavior and error diagnosis than a baseline system, by offering rich meta-information while reducing context switching and mental effort when evaluating multiple MAS conversations. As the first system to link high-level abstractions with low-level MAS details, ConvoMap supports seamless large-scale agent behavior inspection. In summary, this paper contributes:

- A formative study that improves understanding of MAS

1943-6106/25/$31.00 ©2025 IEEE
DOI 10.1109/VL-HCC65237.2025.00043

development's workflow, challenges, and needs.
- A novel technique for encoding MAS conversations as a 2D map and automated qualitative coding of agent messages, addressing a core MAS development challenge: understanding and diagnosing agent behavior at scale.
- Evidence showing that ConvoMap enables more accurate understanding of agents' behavior and error identification in MAS conversations compared to a baseline system.

II. Related Work

This work builds on prior research on multi-agent system development, visualizing code and conversations at scale, and interactive evaluation tools for large language models.

A. Multi-Agent Systems Development

Recent MASs leverage LLMs for flexible communication and autonomous collaboration across diverse tasks [10], [18], [19]. Emerging work highlights the use of LLMs as core agents in MASs [5], [7], [20], supported by frameworks with static collaboration patterns [21], role-based cooperation [22], and agent debates for diverse reasoning [23], [24]. AutoGen is an infrastructure for general MAS development, enabling dynamic conversation patterns, additional tool usage and human input [25]. However, if not designed properly, LLM agents may produce incorrect responses, fail tasks, and generate misleading outputs for other agents [26]. Aligning agents interactions with MAS goals remains challenging. Within this context, platforms like AgentOps [27] aid in testing and debugging AI agents. However, AgentOps's design is oriented towards tracking technical events (e.g., LLM calls, tool usage, API cost and agents' errors) instead of semantic meanings in agents' conversations. AGDebugger supports interactive hypothesis testing for debugging MASs by experimenting with agent messages [17]. Yet, neither AgentOps nor AGDebugger supports large-scale analysis of MAS conversations as ConvoMap does. ConvoMap introduces a novel approach to align agent behavior with MAS goals, by visualizing conversations in a scalable 2D map, providing users with a navigable and efficient way to explore and compare conversations.

Furthermore, little is known about how developers integrate the tools into their workflows or the challenges of MAS development. While Epperson et al. identified difficulty in understanding long agent conversations and iterating on agent configuration [17], their work focused on single-run debugging rather than the cumulative logs from the full development cycle. To address this gap, we investigated real-world MAS development practices and pain points across the entire lifecycle. Our evaluation study presents design considerations for future tools supporting MAS development at scale.

B. Visualizing Code at Scale

Large codebases exhibit similar multi-dimensional complexity as in MAS conversations, including code structure, function calls, dependencies, and incremental changes over time. Code clustering has been widely used to understand the variation among code samples, where similar pieces of code are grouped together to reduce comparison effort [28]–[32]. Many clustering results are static [32] or displayed as a list of code clusters [28], limiting users in inspecting the code content at different levels of detail and tracking incremental changes. VizProg visualizes students' coding progress in real-time by representing their coding status as dynamic dots moving on a 2D map [33]. CFlow visualizes variations among large collections of code samples by organizing them within a shared code framework, based on multiple levels of clustering of code lines [34]. However, there are several key gaps between these designs and the need of understanding conversations in MASs. First, while the 2D map in VizProg provides helpful abstractions and overviews, it does not fully capture the semantic meanings in MAS conversations. Second, CFlow's design is limited to code samples that share similar structures, which is not typical in MAS conversations, where the interactions between agents are often highly varied. ConvoMap adopts the concept of visualizing interactions on a 2D map from VizProg, but augment it with topic summarization. This addition reduces the amount of text-heavy content that users need to read, making the system more efficient for inspecting and understanding MAS conversations.

C. Visualizing Conversations

Various approaches have been proposed to visualize conversations, including the temporal patterns [35], [36], topics and sentiment in conversations [37]–[39], the complex hierarchical replying structures [40], [41], and conversation histories [42]. However, beyond the similarity of sensemaking of conversations, existing conversation visualization solutions have certain limitations. Current tools do no scale well for visualizing and comparing large numbers of conversations. Additionally, one important aspect of MAS conversations is the cooperation among different roles. An error could propagate to other agents and cause other potential problems, which is unique compared to general conversations. ConvoMap is designed as a tool for developers to steer agents' behavior, assess agents' performance, and identify problems in the conversations.

D. Interactive Tools for Evaluating Large Language Models

Researchers have explored ways to incorporate human evaluation into assessment on LLM behaviors to identify flaws and potential improvements [14]–[16]. ChainForge is a low-code framework for comparing responses across models and prompt variations, with customized hypothesis testing [15]. EvalLM is an interactive system that replace manual evaluation with an LLM-based evaluator on user-defined criteria for generated outputs [16]. LLM Comparator is a visual analytics tool for analyzing model results from automatic side-by-side evaluation, with a focus on when and why the responses are qualitatively different across models [14]. However, these systems primarily address single-turn interactions with a single agent, overlooking the complexities of multi-turn, multi-agent systems. Meaningfully evaluating LLMs' performance also remains challenging, as many tasks are domain-specific and require expert judgment. Hybrid approaches combining

qualitative methods with quantitative metrics have been applied to evaluate LLM's outputs [43], including automated assertion generation for error detection [44], [45] and debugging through multi-level prompts [46]. In addition to LLM's outputs, researchers have also developed tools to visualize human's interaction with LLMs. For instance, StuGPTViz tracks and compares temporal patterns in student-ChatGPT interactions [47]. However, these approaches struggle with the complexity of MAS conversations involving multiple agents and diverse message formats. Given that agents' interactions are dynamic and can vary widely, it is difficult to come up with one-size-fits-all assertions for MASs.

III. Interviews with MAS developers

We conducted semi-structured interviews with six industry developers (five men, one woman) to understand their experiences with MAS development, including workflows, challenges, and the information needed for interpreting, debugging, and evaluating MAS outputs. Participants, recruited through screening and snowball sampling at Adobe, had two months to two years of experience with MAS development in industry, and held roles such as research scientists, software developers, and interns. Virtual interviews lasted about an hour. We asked participants how they currently develop MASs, how they monitor agent performance, how they evaluate and understand agent behavior, and what information is most important to them throughout the process. Interviews were recorded, transcribed, and analyzed using thematic analysis. One researcher created an initial coding scheme, and the research team iteratively refined to identify recurring themes and key insights. Consistent with findings from Epperson et al. [17], participants reported difficulties in interpreting long conversations and refining agent configurations. Building on this observation, our study further revealed challenges in evaluating the MAS development lifecycle, coordinating with domain experts, and managing outputs across multiple runs. The full interview protocol is included in the supplemental material.

A. Evaluation Complexity and Bottlenecks

We asked participants to describe how they typically evaluate the outcomes of their MASs, and found that the evaluation process involves both *machine evaluation* and *human evaluation*, which are often performed by different stakeholders. For machine evaluation, developers often handcrafted a set of metrics to programmatically evaluate the outputs, such as similarity to a gold answer, pass rate of test cases, and whether certain keywords were mentioned. While machine evaluation was cheap and quick, it did not fully capture the quality of the outputs. To validate the quality, domain expert collaborators generally conducted a human evaluation to examine the final outputs, which was expensive and slow, but reliable (P1–P6).

The evaluation process was also highly iterative; after implementing new optimizations, a new round of evaluation was often required (P1–P6). Additionally, evaluation was performed on various test cases to cover a broad range of multi-agent system usage scenarios, ensuring the system's consistency.

The time-consuming and iterative nature of the evaluation process make it a critical bottleneck in development. Although intermediate logs were reported to be important for debugging MASs, participants mentioned that they rarely read through them due to the sheer volume and the time required to analyze them (P2, P3, P6). Similarly, while our participants believed tracking the changes across evaluations was important, tracking became nearly "impossible" as numerous conversations were generated (P4). Therefore, they typically monitored the outputs of the most recent runs, focusing on a specific component of the system for the particular problem (P1–P3, P5).

B. Collaborative Gaps in MAS Evaluation

Evaluation and optimization of multi-agent systems requires collective effort from both developers and domain experts. Developers often used intermediate logs to assess agents' functionalities and diagnose failures, or set guardrails during the interactions *between* agents (P1–P4, P6). However, there was no perfect solution that worked 100% of the time due to LLM's randomness (P1–P4, P6). Furthermore, evaluating the overall *quality* of outputs often fell outside the developers' scope due to their lack of domain-specific knowledge.

Domain experts, on the other hand, focused on evaluating the final outputs and provided feedback to developers without considering the intermediate logs generated by agents. Instead, one participant pointed out that someone *"who isn't the developer of that system... will treat the system as a black box"* (P2). This approach generally limited the domain expert's ability to provide feedback on the system's internal functionalities because the domain expert *"won't have the internal knowledge of how... the system works... I am the person developing it, I know exactly how each module and each component functions. Then I will have a more systematic view of the causes of the issue and the associated changes or improvements"* (P2).

This separation between stakeholders and the lack of comprehensive evaluation tools could sometimes lead to miscommunications and inefficiencies in the development process, where developers had to rely on domain experts' feedback to target core problems. Bridging this gap required tools that allowed both developers and domain experts to have a holistic view of the system's performance, including intermediate logs of agent behaviors and the MAS outputs. P3 and P5 described a preliminary exploration to build their own logging systems for comprehensive evaluation by their teams. However, these systems were primarily text-based, limited to internal team use, and struggled to handle multiple, large logs effectively.

C. Challenges of Managing Multiple Runs in MASs

Participants reported having multiple runs of MASs during development for various reasons (P1–P6). For example, to explore various model setups, experiment with test cases from different perspectives, compare slightly different input queries, and test the system's consistency on multiple re-entries for the same query. Participants had between 10 and 100 test cases. For some test cases, they would run re-entries 3-4 times to ensure thorough testing of the system. While the

intermediate logs of a single conversation were not excessively long, the accumulation of logs as the project progressed made tracking conversations difficult (P2, P4–P6). Thus, participants typically focused on key interactions between agents, rather than reading every message and their context (P1–P6). P1 and P2 mentioned the lack of systematic ways to manage the results from multiple runs. Debugging heavily relied on the developer's intuition and knowledge of the MAS. Only when developers were familiar with the system could they reason about which part caused the issue based on the outputs (P2).

D. Limitations with Existing Tools

We also asked participants about the tools they used to develop MASs. Our participants implemented their systems in Python (6/6) and JavaScript (2/6), and typically used frameworks like AutoGen [25] (3/6) and AutoGen studio [48] (1/6) to implement their MASs, CrewAI [49], Huggingface [50], and LangChain [51] for model APIs (3/6), and DSPy [52] for comparing combinations of models (1/6). Participants found these particularly tools helpful in early research stages for their flexibility, maintenance, and usability. However, these tools were often less suitable for production because they have "*too many redundant components... It's inefficient process production, so it's more for research or quickly exploration*" (P2).

In terms of programming environment, participants used VSCode [53] (5/6) and Jupyter Notebook [54] (1/6); while their choice was not driven by specific reasons, the most helpful features were the ability to set breakpoints (P1, P3) and inspect agents' behavior (P1–P3, P5, P6). Jupyter Notebook's interactive nature and intermediate runtime results were beneficial early on, but developers still needed to run the full script at later implementation stage (P3). The current toolset has several limitations: the lack of support for visualizing agents' behavior (P1, P5), assessing individual agents' performance (P2, P3, P5, P6), and debugging tools to identify problems within the system (P1, P5). P1 also raised concerns about the time and effort required to learn a new tools for MAS development.

E. Design Goals

Based on prior work and the experience of our formative study participants, we synthesized three design goals to better address user needs when developing multi-agent systems:

DG1: Provide an intuitive understanding of individual agent behavior across multiple runs. MASs involve multiple black-box agents working in tandem, making it challenging to monitor each agent's contribution. Developers need to understand what each agent is doing, how its behavior evolves across different runs, and whether it fulfills its assigned subtask effectively. While prior work has focused on inspecting agent behavior within individual runs [17], [27], our findings highlight the practical need to manage and compare agent behaviors across multiple runs. To this end, new systems should move beyond raw, text-heavy logs and provide interpretable visualizations that help developers quickly understand, compare, and track agent roles, actions, and outputs over time.

DG2: Facilitate efficient identification and tracing of errors within the agent interactions. Errors in MASs often emerge from complex inter-agent dependencies and can propagate across the system. Developers need to pinpoint where a problem originates in a conversation and how it spreads. The system should help users locate problematic messages, trace the path of interactions leading to the issue, and understand the dependency structure in the logs by reducing the effort needed to read through extensive message histories. While this goal aligns with prior work on tracing individual failures within a single run [17], our goal is to support higher-level diagnosis across multiple runs and agent interactions, enabling developers to detect recurring patterns and systemic issues.

DG3: Enable multi-level exploration and comparison of multi-agent conversations and outcomes. Because MAS outputs vary significantly across runs due to LLM randomness and input variation, developers must compare conversations to identify behavioral patterns and inconsistencies. Extending beyond the needs of debugging MASs [17] and steering task execution [27], the system should support exploration at multiple granularities—zooming into individual messages, examining interactions at the agent or sub-task level, and comparing inputs and outputs across runs—to help developers make sense of emerging patterns and systemic behaviors.

IV. ConvoMap

Guided by our design goals (Section III-E), we developed ConvoMap, a visualization system that enables developers to explore the temporal and structural evolution of agents' conversations, understand agent interactions, identify patterns and errors, and assess which conversations are of low quality. In the following sections, we describe ConvoMap's user interface and the algorithms used to implement its features in detail.

A. ConvoMap's User Interface

ConvoMap's user interface is organized into three panels: a ***Topic Overview*** (Figure 1.1), a ***Semantic Map*** (Figure 1.2), and a ***Conversation Detailed*** View (Figure 1.3). Upon loading conversations from a multi-agent system, ConvoMap determines the semantic meaning, topics, and error information across all messages as described in Section IV-B.

1) Semantic map view shows conversation trajectories: To clearly convey the overall meanings of messages and how conversations evolve (DG1: Individual and DG2: Interactions), ConvoMap represents messages and conversations using dots and trajectories on a 2D map (Figure 1.f). The semantic map view uses a dot to indicate one message from the conversation; the color of the dot indicates which agent the message is from, according to the legend displayed at the top of the semantic map view (Figure 1.h); the size of the dots represent the overall message length. The gray line connecting the dots shows how conversations evolve over time. This temporal information is depicted by left-to-right trajectories, as rightward movement is commonly associated with progress (Figure 1.e).

To illustrate the meanings of messages, the 2D map also encodes semantic similarity between messages as the distance

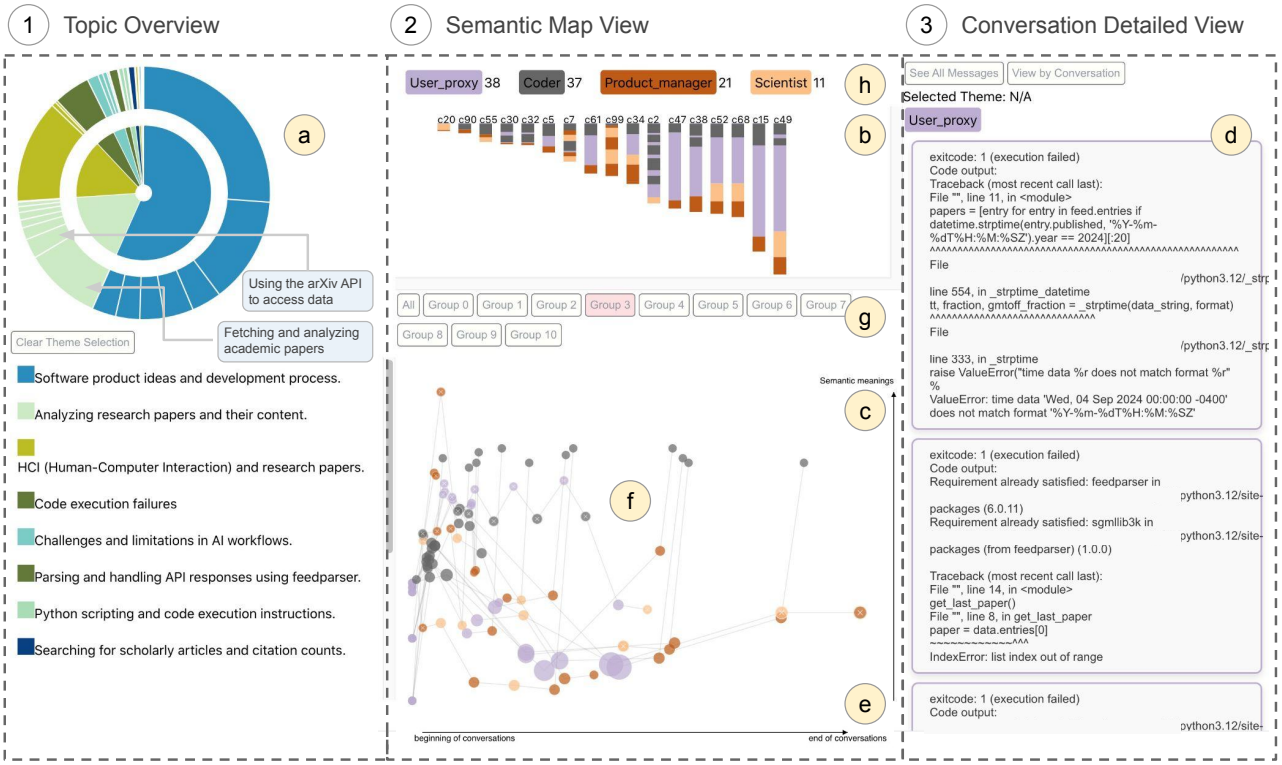

Fig. 1. ConvoMap's user interface displaying the conversation dataset used in our user study. The interface consists of three main views. The ***Topic Overview (1)***, which provides users with donut charts showing the distribution of common topics in the messages (a). The ***Semantic Map View (2)***, which displays agent legends at the top (h), a bar chart of conversations (b), groups of conversations (g), and a 2D map of conversation trajectories (f). The y-axis encodes messages' semantic meanings (c). The x-axis encodes progression of conversations from start to end (e). The ***Conversation Detailed View (3)***, which display all message content in blocks, allowing users to either view by agent or by conversation (d).

on the y-axis (Figure 1.c). Dots that are close together on the y-axis represent messages with similar semantic meanings, while dots that are farther apart indicate messages with different meanings. For instance, the cluster of gray dots in the bottom part of the semantic map in Figure 2.f represent messages saying "TERMINATE", while the purple dots in the upper left corner represent the initial message from the user (Figure 2.g).

Some dots in the semantic map view are marked with an "X" to show that the message may have an error (Figure 2.b). In our implementation, we leverage keyword and text search to detect potential errors, such as messages containing phrases like "execution failed," "misunderstanding," or "unable." This search allows users to quickly identify problematic messages and grasp the meaning of agents' messages at a high level.

To help aid navigation, the semantic map view also groups conversations by the tasks they accomplish (Figure 1.g) and provides a stacked bar chart for each one (Figure 1.b) as an overview. Similar to the 2D map, each segment is color-coded by agent (Figure 1.h), with height representing the message length. Users can interactively click segments or highlight dots in the 2D map (Figure 2.d) to view the messages from an individual conversation. As users select conversations or collections of messages, both the topic overview (Figure 1.1) and the conversation detailed view (Figure 1.3) are updated to reflect this user interaction. Users can select a conversation ID to view its trajectory on the 2D map and see its complete message history in the detailed view (Figure 2.a).

ConvoMap also supports exploring conversation messages at different granularities. Users can examine both groups and individual messages (DG3: Exploration). For groups of messages, users can brush and select a region on the 2D map to focus on the messages within that area (Figure 2.d). Once a region is selected, the area is highlighted with a gray rectangle surrounding multiple dots, and the trajectories passing through that region are highlighted in red (Figure 2.c).

2) Topic overview summarizes key themes in messages: To help users digest the text-heavy content across multiple multi-agent system conversations (DG3: Exploration), ConvoMap summarizes common topics in the conversations and visualizes them in the topic overview (Figure 1.a). This approach is inspired by qualitative coding, where sentences are categorized and summarized into specific themes. We also experimented with word clouds to represent topics but found that they primarily highlighted surface-level keywords, which were insufficient for conveying the nuances of complex conversations.

The topic overview consists of two donut charts: an inner

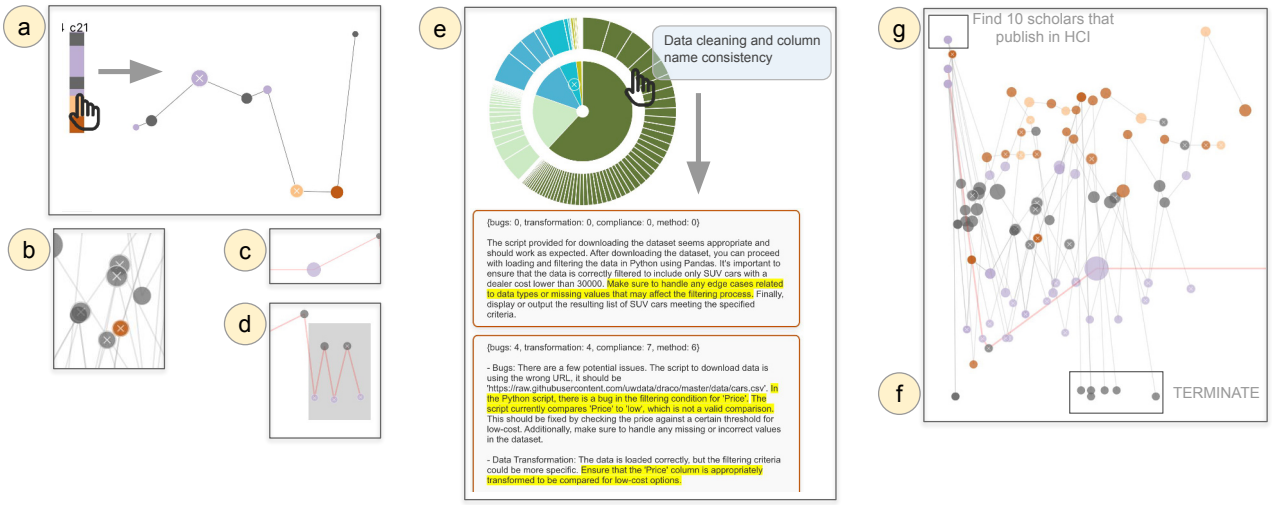

Fig. 2. A detailed view of ConvoMap's user interface allows users to interact with various components. From the ***Semantic Map View***, users can brush and select a region on the 2D map to view messages in that area (d). The selected region appears as a gray rectangle surrounded by dots (d), and the trajectories passing through the selected area are highlighted in red (c). Dots that may contain errors are marked with an "X" (b). Dots that are far apart on y-axis have different meanings (f, g). When a user clicks on a bar chart (a), the corresponding trajectory is highlighted. Additionally, clicking on a topic in the donut chart from the ***Topic Overview*** highlights relevant segments in the textual messages in yellow within the ***Conversation Detailed Views*** (e).

and an outer chart (Figure 1.a). These rings represent two different levels of summarization—the inner chart shows higher-level topics, while the outer chart displays lower-level topics. For each high-level topic in the inner chart (e.g., *"Analyzing research papers and their content"*), there are corresponding lower-level topics in the outer chart (e.g., *"Using the arXiv API to access data"*). High-level topic legends are displayed below the donut charts. We present an example of the topics (Figure 1.a), where the high-level topic represents a general step (as listed in the legend in Figure 1.1), and the two low-level topics represent specific, actionable items to accomplish that step (highlighted by the gray arrows in Figure 1.1).

The topics also function as a search tool to filter messages. When users click on any section of the donut chart, relevant sentences in the conversation are highlighted in yellow, and the detailed conversation view will only display messages related to the selected topic (Figure 2.e).

3) Conversation detailed view shows selected messages: ConvoMap supports multi-granularity exploration by allowing users to select regions on the 2D map and view the actual content in the conversation detailed view. To help users interpret messages, the conversation detailed view offers two modes: (1) "view by agent," where messages are grouped by the agent sender, and (2) "view by conversation," where messages are displayed in the order they appear in the conversation (Figure 1.d). Users can resize regions in the map view or select conversation IDs to view real-time updates across topics and message content in the conversation detailed view.

B. ConvoMap's Algorithm

To generate the aforementioned visualizations, ConvoMap uses three primary processing stages: (1) identify the positions of messages on the 2D map; (2) check whether a message potentially contains an error; and (3) summarize topics from the messages. We describe each stage in the subsections below.

1) Identifying the positions of messages on the 2D map: As discussed in Section IV-A1, we represent the temporal information of conversations on the x-axis and the semantic meaning on the y-axis. Let c denote a given conversation, consisting of a list of k messages $\{m_1, m_2, ...m_k\}$.

Messages' X Positions. For any message m, we define its length, denoted as $len(m)$, to be the number of characters in the message. Compared to using message indices alone to determine x-positions, character count better captures both the turn order and the length of each agent's output. To determine the distance of messages from the beginning of the conversation, we use the cumulative length of messages. For any given message m_i in conversation c, we compute the x-position as the accumulated length from m_1 to m_i in conversaiton c: $x\_position(m_i) = \Sigma_{j=1}^{i} len(m_j)$. Note that $x\_position(m)$ can vary across conversations. When visualizing dots on the 2D map, we apply a linear scaler to ensure that $x\_position(m)$ fits within the width of the map.

Messages' Y Positions. For all messages across conversations in the same group, we first build matrix P containing the vector representation of every item m_i in $\{m_1, m_2, ...m_n\}$, where $vec(m)$ is the vector embedding of a message m, using OpenAI "text-embedding-3-large" embedding model [55]:

$$P = \underset{\in \mathbb{R}^{n \times 3072}}{\begin{bmatrix} | & | & & | \\ vec(m_1) & vec(m_2) & \cdots & vec(m_n) \\ | & | & & | \end{bmatrix}}$$

We reduce P from 3072 rows to 1 row, using T-SNE [56].

This reduces P to a one dimension vector, which we call $\vec{y} =$ T-SNE$(P, 1) \in \mathbb{R}^n$, which we use to compute the y position of each message, denoted as $\vec{y_i} \in \mathbb{R}$ for message m_i.

2) Checking whether a message potentially contains errors: To check whether a message potentially contains an error in the output, we use text search and keyword detection. We collected the following list of keywords for error detection: `"apologies"`, `"apology"`, `"sorry"`, `"confusion"`, `"misunderstanding"`, `"inability"`, `"execution failed"`, `"issue"`, `"unable"`. For a given message m, if it includes any of the keywords, it potentially has an error.

3) Summarizing topics from the messages: We first segment all messages into smaller parts. For natural language messages, we use the sentence tokenizer from NLTK [57] to break them into sentences. For code messages, we split them into smaller code snippets using two consecutive new lines (`"\n\n"`). This method produces high-quality segments, as the code generated by MASs is well-organized and follows a consistent style.

For conversations within the same group, we gather all segments across conversations, denoted as $\{s_1, s_2, ...s_n\}$. Next, we build a matrix $P\_segment$, which contains the vector representation of each item s_i in $\{s_1, s_2, ...s_n\}$, where $\vec{s}$ is the vector embedding of a given segment s, generated using OpenAI "text-embedding-3-large" embedding model [55].

$$P\_segment = \underset{\in \mathbb{R}^{n \times 3072}}{\begin{bmatrix} | & | & & | \\ vec(s_1) & vec(s_2) & \cdots & vec(s_n) \\ | & | & & | \end{bmatrix}}$$

We then apply agglomerative clustering [58] to group segments within each tag τ, ensuring that semantically identical segments are clustered under the same tag. For each cluster, we use GPT-3.5-turbo [59] to generate a natural language description of the topic shared by all segments in the cluster. As a result, for each message m, we obtain a list of topics with their relevant segments: $\{(topic_1, s_1), (topic_2, s_2), ...(topic_k, s_k)\}$. Within message m, we calculate the distribution of topics as the proportion of the segment length, which is used to visualize topics in donut charts:

$$proportion(topic_i) = \frac{len(s_i)}{len(m)}$$

V. User Study

We conduced a within-subjects study with 16 participants to evaluate ConvoMap's efficacy in supporting the exploration and understanding of complex conversations in MASs. The baseline system incorporated core functionalities of currently available commercial tools for monitoring multi-agent systems such as (1) viewing original conversations, (2) grouping messages by agent, and (3) filtering conversations by different tags. In the study, participants used either the baseline system or ConvoMap to answer quiz questions related to agents' behaviors, errors, and patterns. The study was approved by the Institutional Review Board (IRB), ensuring adherence to ethical standards and participant safety.

A. Method

1) Recruitment: We recruited sixteen participants (ten male, six female) from the University of Michigan, all with one to three years of experience working with LLM agents and one to twelve years of Python programming experience. The group included fourteen graduate students, one research engineer, and one postdoctoral researcher. Half of the participants had prior experience working with MAS development, while the other half had limited knowledge about MASs.

2) Study systems: To select a representative baseline and ensure a fair comparison, we first searched for widely used commercial tools designed to monitor MAS behavior. The closest tool was AgentOps [27], which provides a dashboard for visualizing agents' behavior in recorded MAS sessions. However, AgentOps is designed for general inspection tasks, such as tracking LLM calls, event timing, and errors related to LLM API calls, rather than focusing on making text messages more understandable. Additionally, AgentOps accumulates the entire context with every message, which significantly increases the amount of text users read. Moreover, AgentOps' user interface may present a learning curve for participants in the study. To ensure a fair comparison, we implemented our own baseline system, incorporating AgentOps' core features for viewing agents' messages. Our baseline system provides users with a grouped view, allowing messages to be organized by both agents and conversations. Both ConvoMap and the baseline system were implemented as standalone websites.

3) MASs and their conversations: Since no publicly available MAS conversation dataset exists, the first author built two MASs and collected conversations from each. To ensure the authenticity of the data used in the study, the two MASs were adapted from examples on the AutoGen [25] website, a widely used commercial tool for MAS development[1]. For each MAS, we tested 25 prompts with 4 different system settings, collecting up to 100 conversations per MAS. The dataset contains genuine examples of errors and common patterns that agents encounter when modeling real-world tasks, as reported in the formative interviews (Section III). To maintain comparability across conditions, we designed one MAS for each condition with a comparable level of complexity.

Multi-Agent System 1 (MAS-DATA): A data analysis assistant adapted from an automated data visualization example[2]. The MAS first loads an online dataset about car information[3] and writes code to answer user questions about the dataset.

Multi-Agent System 2 (MAS-CHAT): A group chat system that answers research-related questions from users, adapted from an example on automated task solving by group chat[4]. Given a research-related question from the user, the system writes code to provide the answer.

Both MASs had three LLM agents and one user agent, handling user prompts ranging from 5 to 30 words. MAS-DATA

[1]https://microsoft.github.io/autogen/docs/Examples
[2]https://microsoft.github.io/autogen/docs/notebooks/agentchat_groupchat_vis
[3]https://raw.githubusercontent.com/uwdata/draco/master/data/cars.csv
[4]https://microsoft.github.io/autogen/docs/notebooks/agentchat_groupchat

generated 100 conversations, ranging from 6 to 10 messages, with word counts between 70 and 1903. MAS-CHAT generated 100 conversations, ranging from 3 to 12 messages, with word counts between 58 and 3835. In both MAS-DATA and MAS-CHAT, the output messages included three formats: natural language, Python code, and terminal execution results.

4) Study setup: The study was conducted remotely using Zoom using a within-subjects format where participants used both ConvoMap and the baseline system. We counterbalanced the order of the systems and the tasks (MASs). For each condition, we provided 20 minutes of training on how to use the system and the MAS settings, and to read the quiz questions. After training, participants had 20 minutes to answer quiz questions about agents' behavior, errors, and patterns using the assigned system. After finishing the quiz questions, participants were asked to complete a survey about their experience using the system. At the end of each study, we conducted a reflective interview to compare the two systems. We encouraged participants to ask any questions about the usage of both systems. Each session lasted around 90 minutes.

5) Data collection and metrics: During the study, we recorded participants' screens as they completed the tasks, along with their responses to quiz questions, their audio think-aloud processes, and their answers to the post-study survey and follow-up interviews. A member of the research team was present during each study session.

To evaluate participants' answers to the multiple-choice quiz questions, we calculated the F-score as accuracy of each question using the formula:

$$\frac{2 * True\ Positive}{2 * True\ Positive + False\ Positive + False\ Negative}$$

For overall quiz accuracy, we used the following calculation:

$$\frac{sum(accuracy\ of\ all\ questions)}{\#questions}$$

For the open-ended questions, one member in the research team coded participants' answers. We also developed a coding scheme to analyze participants' behaviors in the screen recordings. The screen recordings were coded to analyze the time spent on quiz questions and to understand how participants interacted with both systems to perform the tasks. For the survey responses and follow-up interview data, a thematic analysis was conducted to identify recurring themes and insights. A paired t-test was used for statistical analysis.

B. Results

1) Participants understand agents' behavior more accurately in the ConvoMap condition: We first compared participants' accuracy in answering multiple-choice quiz questions, which required users to locate information about messages and conversations. Accuracy was significantly higher with ConvoMap ($M = 0.84$, $SD = 0.11$) than with the baseline system ($M = 0.70$, $SD = 0.17$, $p < 0.01$). In self-reported survey responses, all participants agreed that ConvoMap better supported them in exploring the conversations than the baseline system ($M = 1.82$, $SD = 0.73$), based on a 7-point likert scale where 1 indicated "ConvoMap" and 7 indicated "ConvoList" (the baseline system). Participants reported significantly higher confidence in using ConvoMap ($M = 4.69$, $SD = 1.40$) compared to the baseline system ($M = 3.81$, $SD = 1.80$) for understanding and diagnosing agents' behavior ($p < 0.05$), based on a 7-point Likert scale where 7 indicated "very confident" and 1 indicated "not confident at all." Participants felt they performed better in understanding messages, agents' interactions, errors, and conversation patterns when using ConvoMap compared to the baseline system. Participants noted that ConvoMap's visualization helped them explore conversations in a more structured way (P1), quickly gain an overview of topics and agent behavior (P1), and reduce the number of messages they needed to read by leveraging meta-information (P3).

2) ConvoMap takes comparable time to explore conversations as the baseline: To assess whether ConvoMap takes more time to explore and understand conversations, we measured the time spent answering the multiple-choice questions. Participants took longer in the ConvoMap condition ($M = 882.63$ seconds, $SD = 180.07$) compared to the baseline condition ($M = 787.00$ seconds, $SD = 143.20$). However, we did not find a statistically significant difference ($p > 0.05$).

Two factors contributed to the comparable times between conditions. First, in the baseline condition, participants had to manually read through every message. All participants started by reading through messages, then gave up midway due to the large amount of text, and moved on to the next question. Second, participants reported that ConvoMap had a learning curve due to its numerous UI components, which could feel overly informative at first. However, participants noted that with more time, they could better navigate the system and explore the conversations effectively (P1, P2, P4, P9, P13, P16).

C. System Usability and Study Insights

1) The semantic map view helps participants quickly make sense of conversations: All participants reported that the semantic map view in ConvoMap helped them quickly make sense of the conversations. From their interview responses, we found that the trajectories connecting the dots in the map provide rich meta-information about the conversations in a coherent and concise view. This visualization reduces context switching and requires less mental and physical effort when exploring conversations. When answering quiz questions and evaluating conversation quality in the ConvoMap condition, fifteen participants first examined the trajectories and dots before diving deeper into the message details.

The semantic map view supports different levels of inspection by allowing various groupings of messages, such as by order, agent, conversation, and similarity. The brushing, selecting, and dragging interactions enable participants to easily filter out irrelevant messages and focus only on the important ones. Furthermore, all types of grouping results can be viewed within a single 2D map, freeing participants from having to manually remember and compare details (P4).

In addition, the semantic map view allows participants to easily identify potential errors in messages and trace their source within a conversation by spotting the "X" marks on the dots. When asked to identify conversations that failed to execute correctly or answer the initial question, ten participants first checked the dots on the map for "X" marks and then examined the detailed message content when using ConvoMap. In contrast, in the baseline condition, participants had to read through all the messages without this additional visual aid.

Trajectories also provided participants with an intuitive visual representation of conversation patterns. For instance, a common pattern was when conversations fell into an infinite loop, where agents repeatedly generated almost identical content without making progress. Rather than reading through all the messages and mentally comparing them as in the baseline condition, participants in the ConvoMap condition were able to quickly recognize zig-zag patterns in the trajectories, indicating that agents were producing similar messages and not progressing. P9 noted that this visual trajectory was more helpful than the flat, linear representation of conversations. The trajectory also provides meta-information, such as agents' engagement and interactions, when agents are communicating with one another (P1, P3-P5, P7-P9, P15, P16).

2) Topic overview supports high-level understanding but can add cognitive load: Participants found the topic overview helpful for understanding the content of agents' conversations (P1, P2, P4, P7, P9, P11, P12, P14-P16). Twelve participants used the topic overview to answer the question, *"Which of the following items are mentioned by the Scientist/Critic agent?"* To ensure a fair comparison, the question topics had a different wording from the exact content displayed in the overview. Participants in the ConvoMap condition had significantly higher accuracy ($M = 0.75$, $SD = 0.15$) compared to the baseline condition ($M = 0.66$, $SD = 0.16$, $p < 0.05$).

Participants noted that the highlighted relevant sentences, when clicking on the topic overview, helped them understand the messages' background (P3, P4, P6, P11, P13). However, they also mentioned that the long list of topics in the overview added to the amount of text they had to read (P4, P6). Also, in some conversations, the topic distribution did not change much, reducing its effectiveness for comparing conversations.

> "I may want to better use this panel [the topic overview]. It could help me identify some of the errors. So I like this high level abstraction. But [...] I would say the details are a little bit [...] overly informative ... It is not that helpful for different sub categories to provide help? I mean half more information for me to diagnose ... But I do like the high level abstraction to give me a brief overview of what are the different topics in this?" (P4)

Furthermore, the topic overview was displayed as a standalone component in ConvoMap's user interface, rather than being integrated into the map or message detail views, leading to extra context switching. While participants found the overview useful for exploring conversations, they did not rely on it much when answering quiz questions targeting specific conversations in a limited time, instead spending more time on the semantic map view and the messages themselves.

3) More context could enhance conversation comparison: In the quiz, we included an open-ended question asking participants to rank four conversations with the same input query from lowest to highest quality. In the ConvoMap condition, fifteen participants first viewed the trajectories in the map view before reading the messages' text, while the remaining participants began by directly reading the messages. To rank the quality, all participants read through the message content. Participants reported difficulty in comparing conversations, as ConvoMap only displayed one conversation at a time, lacking a side-by-side comparison feature (P8, P9, P16).

> "I think that made it a little hard for me to really compare them, because I could only see one at a time. So I have to try and remember in my mind what the differences are, and then also, like I could only see the map for one of them at a time. So I think that made a little hard for me to compare. But I think if the tool kind of showed them side by side, it would be, would have been really helpful." (P9)

Participants explained their reasoning for ranking conversations as high or low quality. We collected 72 reasons across both conditions, with the most common being "failure to answer the question" (19 instances) and "no reasoning and solutions for errors" (8 instances). Both reasons relate to message dependencies within a conversation—specifically, what triggered a message, where it leads, and how it connects to the original query.

4) Future needs: Although we synthesized key user needs in the formative study, we further identified ongoing needs that emerged in the ConvoMap condition of the evaluation study.

The first need is visualizing the dependencies among messages. When asked to identify errors from a specific agent, two participants chose to view the entire conversation rather than focusing solely on that agent's messages, as the context was necessary to understand why the error occurred. Participants noted that while they could identify potential error messages, it was not immediately clear where the error originated (P2, P9).

> "I don't know if either was helping me diagnosing problems in the conversation, because again, the problem itself is hard to quantify ... The problem is not solving the user's query. Then I think both kind of did the same job for me." (P5)

The second need is visualizing variations among conversations at different levels. While ConvoMap allows comparison of conversations through trajectories, participants found this representation too abstract. There was a gap between the abstract trajectory and the detailed text messages, requiring manual effort to remember and compare message content, which led to a lack of clear understanding of the differences.

The third need is to reduce the amount of text users have to read. Ten participants mentioned that reading pure text was time-consuming and difficult to interpret, especially when agents' roles overlapped or the output lacked structure. While

adding more meta-information could aid in understanding, it also risked overwhelming users with too much information.

VI. Discussion

A. Future Design Opportunities

A key challenge in debugging MASs is the lack of guidance and high-level overviews to help developers inspect agent behavior. ConvoMap addresses this gap by enabling exploration of conversations at both high and low levels, helping users identify errors and understand system performance. Our findings show how ConvoMap supports reasoning about complex MAS conversations and reveal unmet user needs. We outline opportunities for future design to better support understanding and debugging of MAS conversations.

1) Visualizing conversation progress through trajectories: While the trajectories help visualize conversation flow, they lack semantic depth. Enhancing trajectories with annotations—such as whether a conversation is progressing or looping, types of errors, and their influence on later messages—could reduce the need to read the text-heavy content. The map could also distinguish content types (e.g., natural language, code, execution results) by distributing them in different regions, and incorporate final outputs or human evaluation results to help users identify patterns in conversation quality.

2) Reducing cognitive load in topic summarization: While topic overviews help users understand message context, they can also increase cognitive load when too many overlapping topics are presented. This issue could be mitigated by allowing users to define their own topics and by refining topic generation parameters. Additionally, the current separation between topic overviews and the conversation view forces users to switch contexts. Integrating topic summaries directly into the semantic map or message view—potentially using alternative representations like avatars—could reduce this effort and support more seamless interpretation.

3) Uncovering agent dependencies in non-linear conversations: When diagnosing errors, participants looked beyond individual messages to understand broader conversational context and dependencies. However, linear message displays made it difficult to trace non-linear flows, such as feedback loops or propagated mistakes. Relationships like revisions or feedback often required referencing distant messages. While trajectory visualizations helped, future designs could better support reasoning by highlighting connections and exploring non-linear message displays.

4) Supporting comparative analysis across conversations: Comparing conversations remains a challenge, yet it is essential for understanding model behavior and performance. Developers need to compare both individual conversations to identify specific errors and groups of conversations to spot broader trends. Future designs could support this task by enabling side-by-side views and integrating comparison tools, such as labeling, within the semantic map to track performance changes across runs or agent settings.

B. Limitations and Future Work

Our user study has three primary limitations. First, the MAS conversation dataset was generated by our research team rather than being sourced from a real-world dataset. This decision may introduce bias in the distribution of input queries and conversation lengths. In a real-world dataset, user queries could exhibit greater variation, adding complexity to understanding conversations. Additionally, the model settings in our MASs may differ from current industry practices. For instance, industry MASs may have stricter rules on maximum message length and employ more accurate mechanisms for speaker selection and task division among agents. Second, our user study did not allow participants to perform actionable tasks, such as modifying the MAS as they read and visualized the conversations. As a result, we lack direct evidence of how ConvoMap would influence the development cycle, particularly in debugging MASs, and its potential impact on the final MAS performance. Third, the participants were primarily graduate students without MAS development experience, making them less familiar with the architecture of such systems. Additionally, there are limitations to the system itself. Several participants noted that both the baseline system and ConvoMap can be overwhelming when viewing conversations at scale. Simplifying the visualization and making it more intuitive remains a challenge. Future work could also investigate how protocols like MCP [60] and A2A [61] are used in practice to provide context to agents and support coordination.

VII. Conclusion

In this work, we conducted six semi-structured interviews to understand existing workflows, challenges, and key information needed for MAS development. We introduced ConvoMap to address a major challenge identified in the interviews: the exploration and diagnosis of multiple, complex conversations in MASs. ConvoMap enables users to inspect large numbers of MAS conversations at multiple levels by presenting messages on a 2D map, augmented with automated qualitative coding to help digest text-heavy content. In ConvoMap, each message is positioned based on its semantic similarity, the progression of the conversation, and how message topics and patterns evolve over time. Our evaluation demonstrated that ConvoMap allows users to more accurately understand agents' behaviors and possible errors compared to the baseline system. Additionally, the 2D map design provides rich meta-information about conversations within a concise view, reducing context switching and the mental effort required to interpret text-heavy messages at scale. This work also offers valuable design insights for understanding and debugging MASs during development.

VIII. Acknowledgments

We would like to thank our formative study participants for sharing their time and expertise around MAS development, as well as our user study participants for their feedback on the design and utility of ConvoMap. We would also like to thank the reviewers of this paper for their valuable feedback.

REFERENCES

[1] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," *Ieee Access*, vol. 6, pp. 28 573–28 593, 2018.

[2] J. Ota, "Multi-agent robot systems as distributed autonomous systems," *Advanced engineering informatics*, vol. 20, no. 1, pp. 59–70, 2006.

[3] D. S. Drew, "Multi-agent systems for search and rescue applications," *Current Robotics Reports*, vol. 2, pp. 189–200, 2021.

[4] A. Lazaridou, A. Potapenko, and O. Tieleman, "Multi-agent communication meets natural language: Synergies between functional and structural language learning," *arXiv preprint arXiv:2005.07064*, 2020.

[5] C. Qian, W. Liu, H. Liu, N. Chen, Y. Dang, J. Li, C. Yang, W. Chen, Y. Su, X. Cong *et al.*, "Chatdev: Communicative agents for software development," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 15 174–15 186.

[6] S. Tisue and U. Wilensky, "Netlogo: Design and implementation of a multi-agent modeling environment," in *Proceedings of agent*, vol. 2004, 2004, pp. 7–9.

[7] J. S. Park, J. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative agents: Interactive simulacra of human behavior," in *Proceedings of the 36th annual acm symposium on user interface software and technology*, 2023, pp. 1–22.

[8] B. Wang, C. Ren, J. Yang, X. Liang, J. Bai, Q.-W. Zhang, Z. Yan, and Z. Li, "Mac-sql: Multi-agent collaboration for text-to-sql," *arXiv preprint arXiv:2312.11242*, 2023.

[9] Z. Yuan, Y. Liu, Y. Cao, W. Sun, H. Jia, R. Chen, Z. Li, B. Lin, L. Yuan, L. He *et al.*, "Mora: Enabling generalist video generation via a multi-agent framework," *arXiv preprint arXiv:2403.13248*, 2024.

[10] Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, M. Zhang, J. Wang, S. Jin, E. Zhou *et al.*, "The rise and potential of large language model based agents: A survey," *Science China Information Sciences*, vol. 68, no. 2, p. 121101, 2025.

[11] A. Smith, *The wealth of nations [1776]*. na, 1937, vol. 11937.

[12] O.-M. Camburu, B. Shillingford, P. Minervini, T. Lukasiewicz, and P. Blunsom, "Make up your mind! adversarial generation of inconsistent natural language explanations," *arXiv preprint arXiv:1910.03065*, 2019.

[13] Y. Elazar, N. Kassner, S. Ravfogel, A. Ravichander, E. Hovy, H. Schütze, and Y. Goldberg, "Measuring and improving consistency in pretrained language models," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 1012–1031, 2021.

[14] M. Kahng, I. Tenney, M. Pushkarna, M. X. Liu, J. Wexler, E. Reif, K. Kallarackal, M. Chang, M. Terry, and L. Dixon, "Llm comparator: Visual analytics for side-by-side evaluation of large language models," in *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, 2024, pp. 1–7.

[15] I. Arawjo, P. Vaithilingam, M. Wattenberg, and E. Glassman, "Chainforge: An open-source visual programming environment for prompt engineering," in *Adjunct Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023, pp. 1–3.

[16] T. S. Kim, Y. Lee, J. Shin, Y.-H. Kim, and J. Kim, "Evallm: Interactive evaluation of large language model prompts on user-defined criteria," in *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 2024, pp. 1–21.

[17] W. Epperson, G. Bansal, V. Dibia, A. Fourney, J. Gerrits, E. Zhu, and S. Amershi, "Interactive debugging and steering of multi-agent ai systems," *arXiv preprint arXiv:2503.02068*, 2025.

[18] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[19] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin *et al.*, "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, no. 6, p. 186345, 2024.

[20] X. Lu and X. Wang, "Generative students: Using llm-simulated student profiles to support question item evaluation," in *Proceedings of the Eleventh ACM Conference on Learning@ Scale*, 2024, pp. 16–27.

[21] "Babyagi," https://github.com/yoheinakajima/babyagi, 2022.

[22] G. Li, H. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem, "Camel: Communicative agents for" mind" exploration of large language model society," *Advances in Neural Information Processing Systems*, vol. 36, pp. 51 991–52 008, 2023.

[23] T. Liang, Z. He, W. Jiao, X. Wang, Y. Wang, R. Wang, Y. Yang, S. Shi, and Z. Tu, "Encouraging divergent thinking in large language models through multi-agent debate," *arXiv preprint arXiv:2305.19118*, 2023.

[24] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch, "Improving factuality and reasoning in language models through multiagent debate," *arXiv preprint arXiv:2305.14325*, 2023.

[25] Q. Wu, G. Bansal, J. Zhang, Y. Wu, S. Zhang, E. Zhu, B. Li, L. Jiang, X. Zhang, and C. Wang, "Autogen: Enabling next-gen llm applications via multi-agent conversation framework," *arXiv preprint arXiv:2308.08155*, 2023.

[26] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 3, pp. 1–45, 2024.

[27] "Agentops," https://www.agentops.ai/, 2022.

[28] E. L. Glassman, J. Scott, R. Singh, P. J. Guo, and R. C. Miller, "Overcode: Visualizing variation in student solutions to programming problems at scale," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 22, no. 2, pp. 1–35, 2015.

[29] H. Yin, J. Moghadam, and A. Fox, "Clustering student programming assignments to multiply instructor leverage," in *Proceedings of the second (2015) ACM conference on learning@ scale*, 2015, pp. 367–372.

[30] A. Luxton-Reilly, P. Denny, D. Kirk, E. Tempero, and S.-Y. Yu, "On the differences between correct student solutions," in *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, 2013, pp. 177–182.

[31] S. Gulwani, I. Radiček, and F. Zuleger, "Automated clustering and program repair for introductory programming assignments," *ACM SIGPLAN Notices*, vol. 53, no. 4, pp. 465–480, 2018.

[32] A. Head, E. Glassman, G. Soares, R. Suzuki, L. Figueredo, L. D'Antoni, and B. Hartmann, "Writing reusable code feedback at scale with mixed-initiative program synthesis," in *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale*, 2017, pp. 89–98.

[33] A. G. Zhang, Y. Chen, and S. Oney, "Vizprog: Identifying misunderstandings by visualizing students' coding progress," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023, pp. 1–16.

[34] A. G. Zhang, X. Tang, S. Oney, and Y. Chen, "Cflow: Supporting semantic flow analysis of students' code in programming problems at scale," in *Proceedings of the Eleventh ACM Conference on Learning@ Scale*, 2024, pp. 188–199.

[35] S. Fu, J. Zhao, H. F. Cheng, H. Zhu, and J. Marlow, "T-cal: Understanding team conversational data with calendar-based visualization," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, ser. CHI '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1–13. [Online]. Available: https://doi.org/10.1145/3173574.3174074

[36] B. Kerr, "Thread arcs: an email thread visualization," *IEEE Symposium on Information Visualization 2003 (IEEE Cat. No.03TH8714)*, pp. 211–218, 2003. [Online]. Available: https://api.semanticscholar.org/CorpusID:16739165

[37] M. El-Assady, V. Gold, C. Acevedo, C. Collins, and D. Keim, "Contovi: Multi-party conversation exploration using topic-space views," in *Computer Graphics Forum*, vol. 35, no. 3. Wiley Online Library, 2016, pp. 431–440.

[38] E. Hoque and G. Carenini, "Multiconvis: A visual text analytics system for exploring a collection of online conversations," in *Proceedings of the 21st international conference on intelligent user interfaces*, 2016, pp. 96–107.

[39] B. C. Kwon, S.-H. Kim, S. Lee, J. Choo, J. Huh, and J. S. Yi, "Visohc: Designing visual analytics for online health communities," *IEEE transactions on visualization and computer graphics*, vol. 22, no. 1, pp. 71–80, 2015.

[40] M. Wattenberg and D. Millen, "Conversation thumbnails for large-scale discussions," in *CHI'03 extended abstracts on Human factors in computing systems*, 2003, pp. 742–743.

[41] K.-P. Yee and M. Hearst, "Content-centered discussion mapping," *Online Deliberation*, 2005.

[42] T. Bergstrom and K. Karahalios, "Conversation clusters: grouping conversation topics through human-computer dialog," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2009, pp. 2349–2352.

[43] Z. Englhardt, R. Li, D. Nissanka, Z. Zhang, G. Narayanswamy, J. Breda, X. Liu, S. Patel, and V. Iyer, "Exploring and characterizing large language models for embedded system development and debugging," in *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, 2024, pp. 1–9.
[44] S. Shankar, H. Li, P. Asawa, M. Hulsebos, Y. Lin, J. Zamfirescu-Pereira, H. Chase, W. Fu-Hinthorn, A. G. Parameswaran, and E. Wu, "Spade: Synthesizing assertions for large language model pipelines," *arXiv preprint arXiv:2401.03038*, 2024.
[45] S. Shankar, J. Zamfirescu-Pereira, B. Hartmann, A. G. Parameswaran, and I. Arawjo, "Who validates the validators? aligning llm-assisted evaluation of llm outputs with human preferences," *arXiv preprint arXiv:2404.12272*, 2024.
[46] T. Wu, E. Jiang, A. Donsbach, J. Gray, A. Molina, M. Terry, and C. J. Cai, "Promptchainer: Chaining large language model prompts through visual programming," in *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, 2022, pp. 1–10.
[47] Z. Chen, J. Wang, M. Xia, K. Shigyo, D. Liu, R. Zhang, and H. Qu, "Stugptviz: A visual analytics approach to understand student-chatgpt interactions," *arXiv preprint arXiv:2407.12423*, 2024.
[48] "Autogen studio," https://microsoft.github.io/autogen/0.2/docs/autogen-studio/getting-started/, 2023.
[49] "Crewai," https://www.crewai.com/, 2023.
[50] "Hugging face," https://huggingface.co, 2016.
[51] "Langchain," https://www.langchain.com/, 2022.
[52] O. Khattab, A. Singhvi, P. Maheshwari, Z. Zhang, K. Santhanam, S. Vardhamanan, S. Haq, A. Sharma, T. T. Joshi, H. Moazam, H. Miller, M. Zaharia, and C. Potts, "Dspy: Compiling declarative language model calls into self-improving pipelines," 2024.
[53] "Visual studio code," https://code.visualstudio.com/, 2016.
[54] "Jupyter notebook," https://jupyter.org, 2016.
[55] "Embeddings," https://platform.openai.com/docs/guides/embeddings, 2025.
[56] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.
[57] "Nltk: Natural language toolkit," https://www.nltk.org/, 2024.
[58] "Agglomerative clustering," https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html, 2007.
[59] "Openai gpt 3.5 turbo," https://platform.openai.com/docs/models/gpt-3-5-turbo, 2023.
[60] "Model context protocol," https://modelcontextprotocol.io/introduction, 2025.
[61] "Agent2agent," https://github.com/google/A2A, 2025.