

# ACE: Moving toward Co-Investigation with the Agentic Code Explorer

Dario Andres Silva Moran<sup>2</sup>, Kristina Brimijoin<sup>1</sup>, Gabriel Enrique Gonzalez<sup>2</sup>,  
Stephanie Houde<sup>1</sup>, Michael Muller<sup>1</sup>, Michelle Brachman<sup>1</sup> and Justin D. Weisz<sup>1</sup>

<sup>1</sup>IBM Research, USA

<sup>2</sup>IBM Research, AR

## Abstract

In this workshop demonstration paper, we present ACE – the Agentic Code Explorer – a prototype agentic system designed to help software developers conduct sensemaking tasks within large code repositories. The design of this system was motivated by the observation that software developers often use AI coding assistants to help understand and ask questions about source code prior to planning and implementing code changes. Using ACE as a testbed, we present initial steps to explore whether a large language model (LLM)-based agent that is capable of invoking external tools and iteratively refining its own outputs (per the agentic design pattern) might be able to robustly support such a code discovery process. In this way, we use ACE as a means to explore more generally how generative models need not solely focus on the artifact production aspects of co-creative tasks; instead they might focus on the *co-investigative* activities where initial understanding and plans are formed.

## Keywords

Agentic AI, Co-creation, Co-investigation, Code understanding, Human-AI collaboration, Mutual theory of mind

## 1. Introduction

Human-AI co-creation is a process by which “humans and AI collaborat[e] on a shared creative product as partners” [1, Abstract]. Explicit in this definition is the focus on a “shared creative product”: the purpose of the co-creative process is to *create*. Commonly, the type of AI model used in such co-creative processes are generative models, specifically because of their ability to produce output artifacts; i.e. their ability to create. A large body of research has demonstrated the value of using generative models to enable co-creative processes (e.g. [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]), and many communities are focused on investigating how best to design co-creative user experiences (e.g. HAI-GEN [14, 15, 16, 17], GenAICHI [18, 19, 20], ICC<sup>1</sup>, HHAI<sup>2</sup>).

Many co-creative use cases involve exploratory steps [21, 22] where the objective of the activity is on the production of knowledge and insight rather than production of artifacts. Indeed, many studies highlight how generative models are valuable for their ability to answer general knowledge questions, explain novel or specialized domains, and help people learn new concepts (e.g. [22, 23, 24, 25, 26, 27]). With the rise of agentic design patterns, in which an AI agent can evaluate and iterate upon its own outputs and invoke external tools, we propose that the time is ripe for these *investigatory* uses of generative models to be brought to the forefront as a core value of generative AI.

In this paper we take an initial step toward focusing on co-investigation as a primary activity with an agentic system via a prototype called ACE – the Agentic Code Explorer. This project was motivated

---

*Joint Proceedings of the ACM IUI Workshops 2025, March 24-27, 2025, Cagliari, Italy*

✉ dario.silva@ibm.com (D. A. Silva Moran); kbrimij@us.ibm.com (K. Brimijoin); gabriel.gonzalez@ibm.com (G. E. Gonzalez); Stephanie.Houde@ibm.com (S. Houde); michael\_muller@us.ibm.com (M. Muller); michelle.brachman@ibm.com (M. Brachman); jweisz@us.ibm.com (J. D. Weisz)

🆔 0000-0002-3049-3139 (D. A. Silva Moran); 0000-0001-5616-9567 (K. Brimijoin); 0009-0001-4818-1205 (G. E. Gonzalez); 0000-0002-0246-2183 (S. Houde); 0000-0001-7860-163X (M. Muller); 0000-0001-8152-441X (M. Brachman); 0000-0003-2228-2398 (J. D. Weisz)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

<sup>1</sup>International Conference on Computational Creativity. <http://computationalcreativity.net/>

<sup>2</sup>Hybrid Human AI Systems for the Social Good. <https://hhai-conference.org/>

by an observation that software developers<sup>3</sup> often use AI coding assistants to help understand and ask questions about source code, in addition to directly producing it [27, 28]. The aspect of co-investigation we have initially designed for occurs at the start of a coding project when a period of exploration and discovery is needed prior to planning and writing new code - particularly when faced with an unfamiliar code repository. After presenting a review of prior work related to the current implementation of ACE and future features planned, we describe the prototype and elaborate on next steps.

Our paper makes the following contributions to the HAI-GEN community:

- We present ACE, an LLM-based agent designed to aid software developers in code sensemaking tasks.
- Through ACE, we take an initial step toward understanding how generative models coupled with agentic capabilities could drive robust co-investigative processes.

## 2. Related work

We review several areas of literature relevant to the current design of ACE (presented in Section 3) including: LLM agents, code understanding needs, and existing AI support tools for software developers. We additionally review literature relevant to mutual theory of mind as background for future work plans described in Section 4.

### 2.1. LLM agents & agentic workflows

Recently, LLMs have been used to drive agentic workflows in which a user specifies a goal and the LLM makes its own determinations of what steps are needed to achieve that goal [29, 30]. These workflows are enabled by two key developments:

1. A pattern in which the LLM is allowed to orchestrate a series of planning, action and assessment steps when formulating a response to a user’s query actions [31, 32, 33].
2. The ability to take actions that invoke tools external to the LLM, such as making API calls, executing software programs, or writing and executing arbitrary code.

Our ACE prototype leverages both of these capabilities within a code environment with the aim of helping users understand unfamiliar code repositories.

### 2.2. Code understanding needs

Researchers have investigated how software developers build mental models of software systems. Often, developers have questions that require links to the broader code base, external information, or interaction with other developers. For example, developers want to know what the implications of a change are [34] or where certain methods are being called [35]. Answering these questions requires knowledge of a code base as a whole. Software developers also ask questions about *why* code was implemented in a certain way. This information may not always be present within the code itself, but may exist externally such as within GitHub issues or external documentation [34, 36]. Developers are also interested in maintaining awareness of their co-workers activities [34], knowing who owns pieces of code [36], and knowing how their colleagues have interacted with their code [37].

Our ACE prototype is designed to provide answers to general questions users can ask about all the code in a single repository. Agentic reasoning is applied to enable the agent to provide helpful answers based on sources like the relevant lines of code and generated code summaries. In future work, described in Section 4, we describe plans to expand the agent’s information sources to Github Issues and developer team profiles.

---

<sup>3</sup>In this paper, the term “software developer” is used broadly to encompass individuals involved in code-related work, including software engineers, architects, and data scientists.

### 2.3. Mutual theory of mind

A new idea emerging in human-centered AI research focuses on improving the quality of human-AI interactions by developing two kinds of models:

- A user’s mental model of the AI agent, and
- The AI agent’s model of the user.

This idea has been described as *mutual theory of mind* [38, 39] because it involves each party forming a model of the other party’s (mental) state, such as their knowledge, beliefs, intentions, and perceptions. We believe mutual theory of mind will improve the quality of human-AI interactions because (1) it encourages the design of features that help users form mental models of how to effectively use an AI system (e.g. by understanding its capabilities and limitations), and (2) it enables an AI system to interact with a user on an individual level (e.g. by tailoring how it responds to that particular user’s needs, preferences, and level of expertise). Weisz et al. provide additional future scenarios for how mutual theory of mind can improve (or not) the quality of human-AI interactions.

As an initial step toward operationalizing mutual theory of mind, our ACE prototype modifies agent responses with prompts derived from an editable profile representing the user’s expertise (Figure 2). In future work, we plan to enrich this feature to facilitate proactive support by the agent when observation of user behavior and questions suggest this would be useful, as well as providing profile information about the agent that the user could use to inform their questions.

### 2.4. AI support tools for software developers

Researchers have validated the usefulness of generative AI for assisting software developers [9, 41, 42] and for helping students learning programming [43]. Such assistance need not be limited to acts of code production: a recent survey of software developers showed that *understanding code* was a top use for an AI software engineering assistant [27]. However, opportunities remain to improve support for code understanding, such as including user and code context [37, 44], as well as connecting code with external information [45].

A number of commercially available tools offer support for code understanding tasks, including Cursor<sup>4</sup>, Claude<sup>5</sup>, and GitHub Copilot<sup>6</sup>. Our current ACE prototype offers similar code understanding support in a form that we can use to freely build experimental functional and user experience enhancements. In future work, we aim to enrich ACE to explore whether we can provide higher quality personalized support for code understanding through prompting the AI with a more holistic understanding of the the user, their goals, and team context.

## 3. ACE: the Agentic Code Explorer for investigating code repositories

We developed the Agentic Code Explorer (ACE) to explore how agentic design patterns could enable richer forms of assistance beyond mere LLM-based text generation. Our initial prototype was built as a standalone web application, rather than as a plugin for existing developer IDEs, to allow us the freedom to control all aspects of interaction design, model prompting, agentic tool design, interaction design, and usage data collection during the early stages of our research. Recognizing that developers often prefer having new development tools integrated into their IDE [9], we anticipate future versions of ACE to be implemented as such.

---

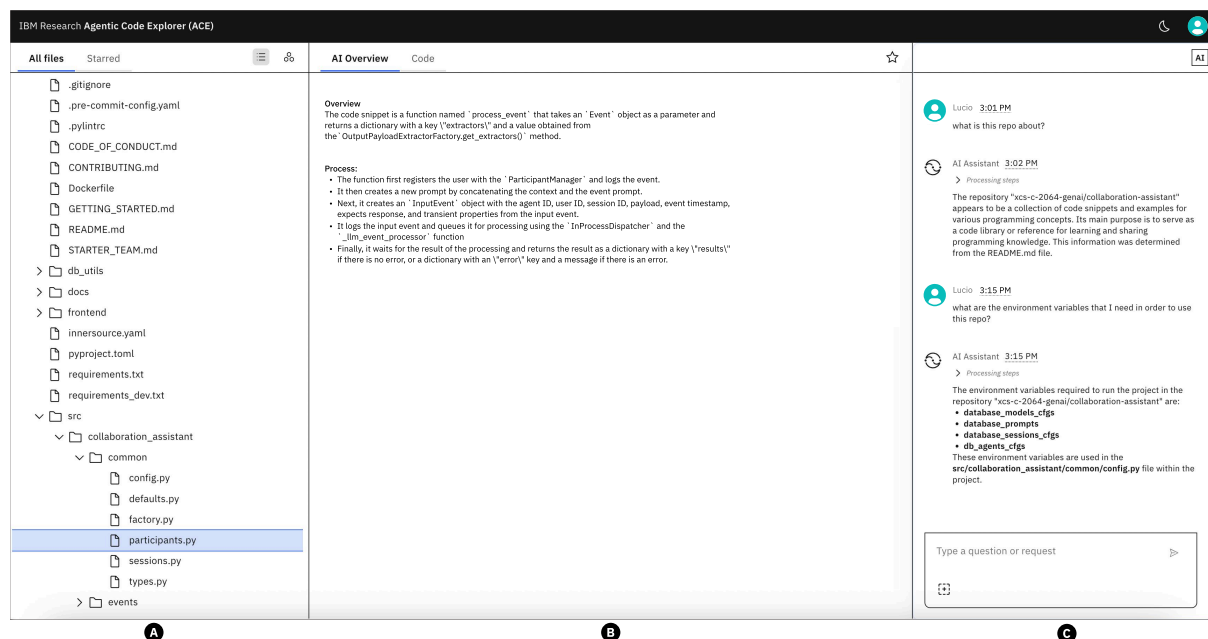
<sup>4</sup>Cursor. <https://www.cursor.com>

<sup>5</sup>Claude. <https://www.anthropic.com/claude>

<sup>6</sup>GitHub Copilot. <https://github.com/features/copilot>

### 3.1. ACE interface

We show ACE’s interface in Figure 1. ACE’s front end was built using the Svelte framework<sup>7</sup> and its back end was implemented in Python using FastAPI<sup>8</sup>.



**Figure 1: ACE User Interface.** Three interface panels allow the user to (A) browse all files in a GitHub code repository, (B) view either the raw source code or an AI-generated summary of a selected code file (shown above), and (C) converse with an LLM-based agent that has access to the full code repository and can answer general programming questions and targeted questions about a file, a code summary, or selected lines of code.

ACE presents a code repository’s files in a tree structure (Figure 1A), mirroring how they are typically presented in developer IDEs. When a file is selected, ACE generates a natural language overview of the file in the central panel (Figure 1B). The overview provides a high-level description of the file’s functionality, summarizing the purpose of the code and all of the classes and methods defined in it. Within the same panel, the user can switch to the “Code” view to see the raw source code.

Users can converse with ACE in the right sidebar (Figure 1C). ACE uses the user’s current context – which file they have open and any text they have selected in the interface – to answer questions. Users can also select text in the interface and press a keyboard shortcut to open a chat input, allowing them to ask ACE a question about anything they see on the screen.

As ACE uses an agentic workflow, the latency of its responses may be high. Therefore, ACE delivers its responses to the chat UI via a streaming API, enabling the user to follow its progress as it processes the query (Figure 3). Users can view or hide ACE’s processing steps, which show information about ACE’s reasoning, observations, tool invocations and outputs.

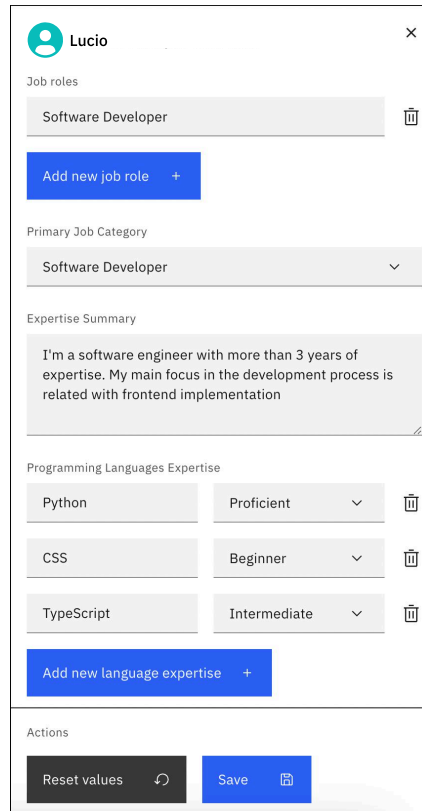
### 3.2. Agentic workflow design

ACE is a conversational agent built using the LangGraph<sup>9</sup> framework. We chose LangGraph for its ability to provide fine-grained control over workflow execution, built-in persistence, and robust LLM integration [46, 47]. ACE operates as a ReAct-based agent [31] composed of four interconnected nodes: Planner, Agent, Tools, and Observer. Each node has specific responsibilities and is guided by a custom prompt template, shown in Appendices A, B, and C.

<sup>7</sup>Svelte. <https://svelte.dev>

<sup>8</sup>FastAPI. <https://fastapi.tiangolo.com>

<sup>9</sup>LangGraph. <https://langchain-ai.github.io/langgraph/>



The image shows a user profile form for a user named Lucio. The form is organized into several sections:

- Job roles:** A list containing 'Software Developer' with a trash icon to its right. Below the list is a blue button labeled 'Add new job role +'.
- Primary Job Category:** A dropdown menu currently showing 'Software Developer' with a downward arrow.
- Expertise Summary:** A text area containing the text: 'I'm a software engineer with more than 3 years of expertise. My main focus in the development process is related with frontend implementation'. There is a small icon in the bottom right corner of the text area.
- Programming Languages Expertise:** A section with three rows. Each row has a language name in a box and a proficiency level in a dropdown menu, with a trash icon to the right of each dropdown.
 

Language	Proficiency
Python	Proficient
CSS	Beginner
TypeScript	Intermediate

 Below this section is a blue button labeled 'Add new language expertise +'.
- Actions:** A section at the bottom with two buttons: 'Reset values' (with a circular arrow icon) and 'Save' (with a floppy disk icon).

**Figure 2: User Profile.** Information in this profile is used by ACE to adapt its responses to the user. For this user, ACE will provide more direct and succinct responses to questions about Python, and longer and more explanatory responses to questions about CSS.

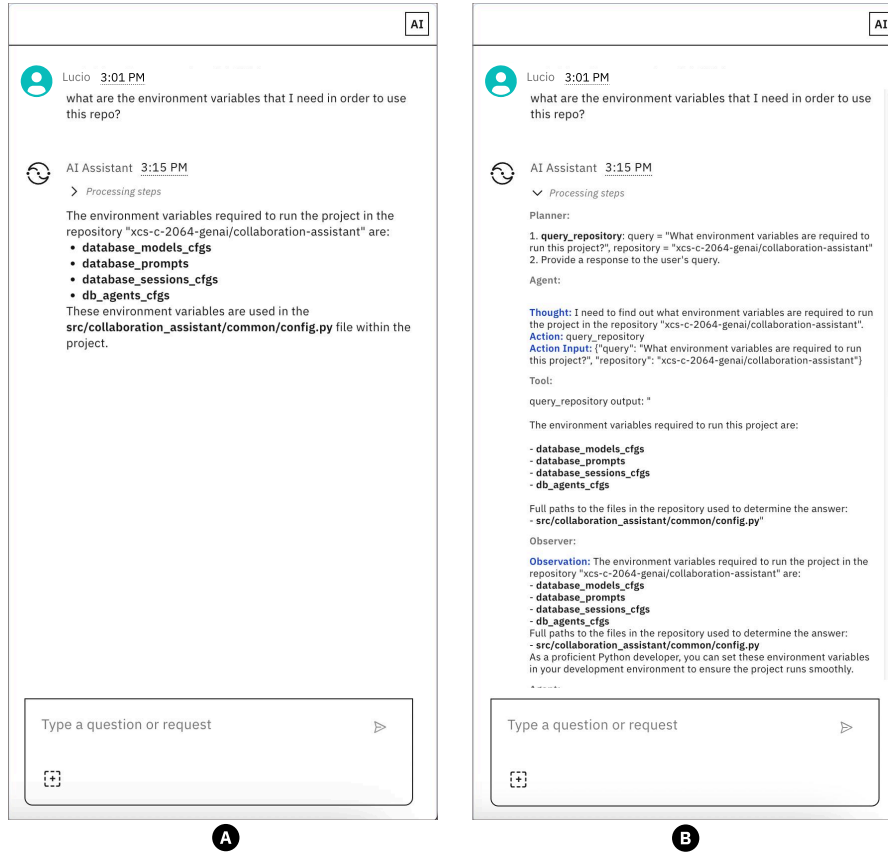
We equipped ACE with tools tailored for summarizing and explaining code (Figure 1B), understanding code dependencies and querying the repository (Figure 3), and visualizing function calls (Figure 4). Since these features operate at the level of a code repository, ACE first performs a preprocessing stage that divides the repository into manageable chunks using LangChain’s `CodeTextSplitter`, embeds them using the `all-minilm-16-v2` model, and stores the resulting vectors in an ElasticSearch database. This setup enables ACE to perform retrieval-augmented generation (RAG) [48] in response to a user’s query.

For code summarization, the agent pre-generates natural language summaries at multiple granularities: one for each function or method within the repository, one for each class, and one for each source file. ACE uses the `code llama-34b-instruct` model to produce these summaries.

## 4. Opportunities & next steps

We see two broad opportunities to improve ACE’s ability to provide useful assistance to software developers. The first involves increasing the amount of information ACE has about the code itself and developing more tools to digest, organize, present that information to users. For example, ACE could incorporate multiple linked code repositories as well as information about code rationale embedded in GitHub Issues. It could also generate documentation that is missing from the repository or update outdated documentation. It can be equipped with more tools that enable it to render this information in new and different visual formats, akin to systems that generate visualizations from natural language [49, 50]. It can also be more tightly-integrated within a developer’s IDE rather than existing as a standalone tool.

The second opportunity stems from recognizing that individuals will differ in the types of support they will need from an AI agent. We can build capabilities into ACE that (1) help users understand



**Figure 3: Agentic workflow.** When a user asks a question, ACE executes an agentic workflow to produce a response (A). Users can examine the process used by ACE to generate that response by viewing its thoughts, actions, tool invocations, and outputs (B).

what kinds of assistance ACE is capable of and (2) allow ACE to recognize those individual differences and tailor its assistance on an individual basis. This idea goes beyond mere personalization because it involves aligning two models – the human’s mental model of ACE and ACE’s model of the user and their mental state. This idea has recently been introduced within the human-centered AI community as mutual theory of mind [38, 39], and prior work has examined how operationalizing it can be beneficial in various work domains [40].

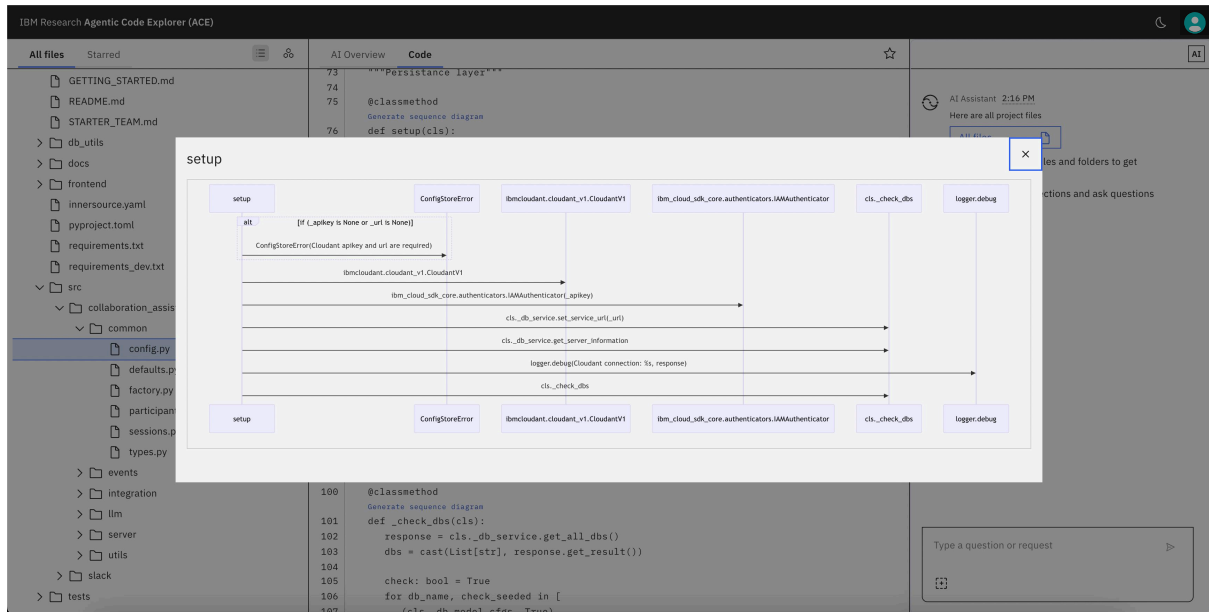
We aim to build on the mutual theory of mind model of Wang and Goel [38]. They described a three-step process in which (1) an AI agent computes an initial model of the human; (2) the human may discover that the model is inaccurate, and may then ask the AI to revise that model; (3) the AI revises its model. Wang and Goel [38] primarily addressed the traits of the user (e.g., “Liz seems to enjoy artistic activities”). We propose two extensions of this model. First, ACE would combine both the ability to maintain a user model of *traits* (e.g., Figure 2) with a more dynamic user model of the human’s current goal or *states* (e.g., Figure 3). These state-oriented activities could be carried out through a fine-grained form of agentic patterns [31, 32, 33].

The second extension under consideration is an expansion of ideas that were hinted at by Wang and Goel [38]. Their model focused on how the AI builds a model of the user. In a complementary way, ACE has the potential to assist the user in building a mental model of ACE and its capabilities. This process would be a role-reversal of the three-step process in the preceding paragraph, and might involve iterative approaches to Explainable AI (XAI) [51].

Within the context of ACE, an operationalized mutual theory of mind would enable ACE to provide new forms of support:

- **Infer code purpose and intent.** Frequently, the motivations for *why* a block of code exists are not well documented. We believe ACE can piece together the context needed to answer such





**Figure 4: Visualizing functions.** ACE is able to invoke tools that produce visualizations as outputs, such as this diagram showing a sequence of function calls.

questions about motivation by examining repository metadata, such as team discussions, GitHub Issues and commit histories. We also believe ACE could capture and record such motivations *at the moment code is written* and then recall them in the future when they are needed.

- **Facilitate team coordination by understanding team expertise.** Software development happens within teams and teams that have a high degree of shared knowledge of team member skills, task knowledge, and current activities [52]. We believe ACE could facilitate team coordination by developing a similarly deep understanding of team member expertise. It could do this by examining relevant data (i.e. code) and metadata (i.e. comments made on issues or pull requests, chat messages, etc.), as well as by asking users to self-describe their expertise (i.e. akin to our profile in Figure 2). Once equipped with such profiles, ACE would no longer need to attempt to answer every user query itself; rather, it could refer to other people on the team who possess the relevant expertise or experience. In this way, ACE may provide some of the human connections that deepen collective knowledge (e.g., distributed cognition [53]) in diverse teams [54, 55].
- **Align agent responses to individual users.** With a model of an individual user’s expertise, ACE could align the way it responds to that user in a more tailored way. For example, a developer who is less familiar with a specific programming language can be provided with code explanations that fill in syntactical gaps, whereas a developer having that expertise will receive an explanation that omits language-specific guidance.
- **Act proactively.** ACE currently provides assistance in a reactive fashion: when the user makes a query. But if ACE has a deep understanding of the user’s *intent* and *goals* – e.g. *why* they need to understand a particular block of code – it may be able to offer assistance in a proactive fashion. For example, a user may need to understand where in a large repository to add new functionality. Rather than providing summaries of files as requested by the user, ACE may be able to proactively suggest which files to look at and offer suggestions for where the new code should be added. Taken to the limit, ACE might be able to organize the whole software development process, aided through discussions with the developer to ensure it maintains alignment with their intent.

We believe such features will enable ACE to be an effective collaborator within human-AI teams, leading to synergistic team outcomes [56]. Thus, as we build these features, we recognize the importance of evaluating their efficacy with users early and often.

## 5. Conclusion

We present ACE, a prototype Agentic Code Explorer that aids software developers in understanding code repositories. ACE was designed as a testbed for examining new approaches to human-AI collaboration for co-investigative tasks with a specific focus on incorporating features that help users establish a mutual theory of mind with the agent. It uses the ReAct paradigm to plan its response to a user query and it invokes tools that either retrieve information from the code repository or generate new representations of that information. Although the primary focus of our prototype has been on enabling the agentic workflow, future development of ACE will focus on how it can achieve a strong degree of alignment with its users' needs and intentions. In this way, ACE represents an initial step toward understanding how generative models need not solely focus on the production aspects of co-creative tasks; instead, agentic design patterns may enable generative models to act as robust information supports for users engaged in co-investigation tasks such as understanding for purposes of development task planning and completion.

## References

- [1] J. Rezwana, M. L. Maher, Designing creative ai partners with cofi: A framework for modeling interaction in human-ai co-creative systems, *ACM Transactions on Computer-Human Interaction* 30 (2023) 1–28.
- [2] X. Du, P. An, J. Leung, A. Li, L. E. Chapman, J. Zhao, Deepthink: Designing and probing human-ai co-creation in digital art therapy, *International Journal of Human-Computer Studies* 181 (2024) 103139.
- [3] S. Wang, S. Menon, T. Long, K. Henderson, D. Li, K. Crowston, M. Hansen, J. V. Nickerson, L. B. Chilton, Reelframer: Human-ai co-creation for news-to-video translation, in: *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 2024, pp. 1–20.
- [4] S. Suh, M. Chen, B. Min, T. J.-J. Li, H. Xia, Luminate: Structured generation and exploration of design space with large language models for human-ai co-creation, in: *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 2024, pp. 1–26.
- [5] S. Houde, S. I. Ross, M. Muller, M. Agarwal, F. Martinez, J. T. Richards, K. Talamadupula, J. D. Weisz, Opportunities for generative AI in UX modernization 81–91, in: A. Smith-Renner, O. Amir (Eds.), *Joint Proceedings of the IUI 2022 Workshops: APEX-UI, HAI-GEN, HEALTHI, HUMANIZE, TExSS, SOCIALIZE co-located with the ACM International Conference on Intelligent User Interfaces (IUI 2022)*, Virtual Event, Helsinki, Finland, March 21–22, 2022, volume 3124 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 81–91.
- [6] D. Yang, Y. Zhou, Z. Zhang, T. J.-J. Li, R. LC, Ai as an active writer: Interaction strategies with generated text in human-ai collaborative fiction writing, in: A. Smith-Renner, O. Amir (Eds.), *Joint Proceedings of the IUI 2022 Workshops: APEX-UI, HAI-GEN, HEALTHI, HUMANIZE, TExSS, SOCIALIZE co-located with the ACM International Conference on Intelligent User Interfaces (IUI 2022)*, Virtual Event, Helsinki, Finland, March 21–22, 2022, volume 3124 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 56–65.
- [7] J. D. Weisz, M. Muller, S. Houde, J. Richards, S. I. Ross, F. Martinez, M. Agarwal, K. Talamadupula, Perfection not required? human-ai partnerships in code translation, in: *Proceedings of the 26th International Conference on Intelligent User Interfaces*, 2021, pp. 402–412.
- [8] J. D. Weisz, M. Muller, S. I. Ross, F. Martinez, S. Houde, M. Agarwal, K. Talamadupula, J. T. Richards, Better together? an evaluation of ai-supported code translation, in: *Proceedings of the 27th International Conference on Intelligent User Interfaces*, 2022, pp. 369–391.
- [9] S. I. Ross, F. Martinez, S. Houde, M. Muller, J. D. Weisz, The programmer's assistant: Conversational interaction with a large language model for software development, in: *Proceedings of the 28th International Conference on Intelligent User Interfaces*, 2023, pp. 491–514.



- [10] D. Bau, H. Strobelt, W. Peebles, J. Wulff, B. Zhou, J.-Y. Zhu, A. Torralba, Semantic photo manipulation with a generative image prior, arXiv preprint arXiv:2005.07727 (2020).
- [11] M. Muller, S. Houde, G. Gonzalez, K. Brimijoin, S. I. Ross, D. A. S. Moran, J. D. Weisz, Group brainstorming with an ai agent: Creating and selecting ideas, in: International Conference on Computational Creativity, 2024.
- [12] O. Shaer, A. Cooper, A. L. Kun, O. Mokryn, Toward enhancing ideation through collaborative group-ai brainwriting., in: A. Soto, E. Zangerle (Eds.), Joint Proceedings of the ACM IUI 2024 Workshops co-located with the 29th Annual ACM Conference on Intelligent User Interfaces (IUI 2024), Greenville, South Carolina, USA, March 18, 2024, volume 3660 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024.
- [13] B. Tilekbay, S. Yang, M. A. Lewkowicz, A. Suryapranata, J. Kim, Expressedit: Video editing with natural language and sketching, in: Proceedings of the 29th International Conference on Intelligent User Interfaces, 2024, pp. 515–536.
- [14] W. Geyer, L. B. Chilton, J. D. Weisz, M. L. Maher, Hai-gen 2021: 2nd workshop on human-ai co-creation with generative models, in: Companion Proceedings of the 26th International Conference on Intelligent User Interfaces, 2021, pp. 15–17.
- [15] J. D. Weisz, M. L. Maher, H. Strobelt, L. B. Chilton, D. Bau, W. Geyer, Hai-gen 2022: 3rd workshop on human-ai co-creation with generative models, in: Companion Proceedings of the 27th International Conference on Intelligent User Interfaces, 2022, pp. 4–6.
- [16] M. L. Maher, J. D. Weisz, L. B. Chilton, W. Geyer, H. Strobelt, Hai-gen 2023: 4th workshop on human-ai co-creation with generative models, in: Companion Proceedings of the 28th International Conference on Intelligent User Interfaces, 2023, pp. 190–192.
- [17] W. Geyer, M. L. Maher, J. D. Weisz, D. Buschek, L. B. Chilton, Hai-gen 2024: 5th workshop on human-ai co-creation with generative models, in: Companion Proceedings of the 29th International Conference on Intelligent User Interfaces, 2024, pp. 122–124.
- [18] M. Muller, L. B. Chilton, A. Kantosalo, C. P. Martin, G. Walsh, Genaichi: generative ai and hci, in: CHI conference on human factors in computing systems extended abstracts, 2022, pp. 1–7.
- [19] M. Muller, L. B. Chilton, A. Kantosalo, Q. V. Liao, M. L. Maher, C. P. Martin, G. Walsh, Genaichi 2023: Generative ai and hci at chi 2023, in: Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems, CHI EA '23, Association for Computing Machinery, New York, NY, USA, 2023. URL: <https://doi.org/10.1145/3544549.3573794>. doi:10.1145/3544549.3573794.
- [20] M. Muller, A. Kantosalo, M. L. Maher, C. P. Martin, G. Walsh, Genaichi 2024: Generative ai and hci at chi 2024, in: Extended Abstracts of the CHI Conference on Human Factors in Computing Systems, 2024, pp. 1–7.
- [21] M. Kreminski, I. Karth, M. Mateas, N. Wardrip-Fruin, Evaluating mixed-initiative creative interfaces via expressive range coverage analysis., in: A. Smith-Renner, O. Amir (Eds.), Joint Proceedings of the IUI 2022 Workshops: APEX-UI, HAI-GEN, HEALTHI, HUMANIZE, TExSS, SOCIALIZE co-located with the ACM International Conference on Intelligent User Interfaces (IUI 2022), Virtual Event, Helsinki, Finland, March 21-22, 2022, volume 3124 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 34–45.
- [22] J. D. Weisz, J. He, M. Muller, G. Hoefer, R. Miles, W. Geyer, Design principles for generative ai applications, in: Proceedings of the CHI Conference on Human Factors in Computing Systems, 2024, pp. 1–22.
- [23] M.-K. Ghali, Y. Jin, Empowering research: Open-source llms, semantic search, and domain-specific knowledge in a multi-document q&a assistant, in: IISE Annual Conference. Proceedings, Institute of Industrial and Systems Engineers (IISE), 2024, pp. 1–6.
- [24] M. A. Arefeen, B. Debnath, S. Chakradhar, Leancontext: Cost-efficient domain-specific question answering using llms, *Natural Language Processing Journal* 7 (2024) 100065.
- [25] F. Yang, P. Zhao, Z. Wang, L. Wang, J. Zhang, M. Garg, Q. Lin, S. Rajmohan, D. Zhang, Empower large language model to perform better on industrial domain-specific question answering, arXiv preprint arXiv:2305.11541 (2023).
- [26] Y. Zhang, L. Chen, S. Li, N. Cao, Y. Shi, J. Ding, Z. Qu, P. Zhou, Y. Bai, Way to specialist: Closing loop

between specialized llm and evolving domain knowledge graph, arXiv preprint arXiv:2411.19064 (2024).

- [27] J. D. Weisz, S. Kumar, M. Muller, K.-E. Browne, A. Goldberg, E. Heintze, S. Bajpai, Examining the use and impact of an ai code assistant on developer productivity and experience in the enterprise, arXiv preprint arXiv:2412.06603 (2024).
- [28] A. Ghimire, J. Edwards, Coding with ai: How are tools like chatgpt being used by students in foundational programming courses, in: International Conference on Artificial Intelligence in Education, Springer, 2024, pp. 259–267.
- [29] M. Mitchell, A. Ghosh, S. Luccioni, G. Pistilli, Ai agents are here. what now?, HuggingFace Blog, Articles, and discussions (2025). URL: <https://huggingface.co/blog/ethics-soc-7>.
- [30] A. Gutowska, What are ai agents?, IBM Think: Tech news, education and events (2024). URL: <https://www.ibm.com/think/topics/ai-agents>.
- [31] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, Y. Cao, React: Synergizing reasoning and acting in language models, 2023. URL: <https://arxiv.org/abs/2210.03629>. arXiv: 2210.03629.
- [32] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, S. Yao, Reflexion: Language agents with verbal reinforcement learning, Advances in Neural Information Processing Systems 36 (2024).
- [33] B. Xu, Z. Peng, B. Lei, S. Mukherjee, Y. Liu, D. Xu, Rewoo: Decoupling reasoning from observations for efficient augmented language models, arXiv preprint arXiv:2305.18323 (2023).
- [34] A. J. Ko, R. DeLine, G. Venolia, Information needs in collocated software development teams, in: 29th International Conference on Software Engineering (ICSE’07), IEEE, 2007, pp. 344–353.
- [35] J. Sillito, G. C. Murphy, K. De Volder, Questions programmers ask during software evolution tasks, in: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering, 2006, pp. 23–34.
- [36] T. D. LaToza, G. Venolia, R. DeLine, Maintaining mental models: a study of developer work habits, in: Proceedings of the 28th International Conference on Software Engineering, ICSE ’06, Association for Computing Machinery, 2006, p. 492–501.
- [37] J. Richards, M. Wessel, What you need is what you get: Theory of mind for an llm-based code understanding assistant, arXiv preprint arXiv:2408.04477 (2024).
- [38] Q. Wang, A. K. Goel, Mutual theory of mind for human-ai communication, arXiv preprint arXiv:2210.03842 (2022).
- [39] Q. Wang, S. Walsh, M. Si, J. Kephart, J. D. Weisz, A. K. Goel, Theory of mind in human-ai interaction, in: Extended Abstracts of the CHI Conference on Human Factors in Computing Systems, 2024, pp. 1–6.
- [40] J. D. Weisz, M. Muller, A. Goldberg, D. A. S. Moran, Expedient assistance and consequential misunderstanding: Envisioning an operationalized mutual theory of mind, arXiv preprint arXiv:2406.11946 (2024).
- [41] J. T. Liang, C. Yang, B. A. Myers, A large-scale survey on the usability of ai programming assistants: Successes and challenges, in: Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, 2024, pp. 1–13.
- [42] R. Khojah, M. Mohamad, P. Leitner, F. G. de Oliveira Neto, Beyond code generation: An observational study of chatgpt usage in software engineering practice, Proceedings of the ACM on Software Engineering 1 (2024) 1819–1840.
- [43] M. Liffiton, B. E. Sheese, J. Savelka, P. Denny, Codehelp: Using large language models with guardrails for scalable support in programming classes, in: Proceedings of the 23rd Koli Calling International Conference on Computing Education Research, 2023, pp. 1–11.
- [44] D. Nam, A. Macvean, V. Hellendoorn, B. Vasilescu, B. Myers, Using an llm to help with code understanding, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, ICSE ’24, Association for Computing Machinery, 2024.
- [45] K. Reefman, Using LLMs to aid developers with code comprehension in codebases, Master’s thesis, University of Twente, 2024.
- [46] J. Wang, Z. Duan, Agent ai with langgraph: A modular framework for enhancing machine translation using large language models, 2024. URL: <https://arxiv.org/abs/2412.03801>.

arXiv:2412.03801.

- [47] C. Jeong, A study on the implementation method of an agent-based advanced rag system using graph, 2024. URL: <https://arXiv.org/abs/2407.19994>.
- [48] J. Chen, R. Bao, H. Zheng, Z. Qi, J. Wei, J. Hu, Optimizing retrieval-augmented generation with elasticsearch for enhanced question-answering systems, 2024. URL: <https://arxiv.org/abs/2410.14167>. arXiv:2410.14167.
- [49] P. Maddigan, T. Susnjak, Chat2vis: generating data visualizations via natural language using chatgpt, codex and gpt-3 large language models, *Ieee Access* 11 (2023) 45181–45193.
- [50] G. Li, X. Wang, G. Aodeng, S. Zheng, Y. Zhang, C. Ou, S. Wang, C. H. Liu, Visualization generation with large language models: An evaluation, arXiv preprint arXiv:2401.11255 (2024).
- [51] J. Y. Bo, P. Hao, B. Y. Lim, Incremental xai: Memorable understanding of ai with incremental explanations, in: *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 2024, pp. 1–17.
- [52] J. A. Espinosa, S. A. Slaughter, R. E. Kraut, J. D. Herbsleb, Team knowledge and coordination in geographically distributed software development, *Journal of management information systems* 24 (2007) 135–169.
- [53] V. Mancuso, Distributed teams in the living laboratory: Applications of transactive memory, in: *Cognitive Systems Engineering*, CRC Press, 2017, pp. 47–65.
- [54] J. Kotlarsky, B. van den Hooff, L. Houtman, Are we on the same page? knowledge boundaries and transactive memory system development in cross-functional teams, *Communication research* 42 (2015) 319–344.
- [55] Y. Wang, Q. Huang, R. M. Davison, F. Yang, Effect of transactive memory systems on team performance mediated by knowledge transfer, *International Journal of Information Management* 41 (2018) 65–79.
- [56] M. Vaccaro, A. Almaatouq, T. Malone, When combinations of humans and ai are useful: A systematic review and meta-analysis, *Nature Human Behaviour* (2024) 1–11.

## A. Planning Prompt

```
You are a helpful AI Planning Agent designed to provide a high-level plan for a Human to
complete a task.
You will be provided a set of [TOOLS] that the Human can use to complete the task.
Each step of the plan should involve a tool call from the [TOOLS] provided or a response to
the user.
Do not add a step if it is not a tool call or if it is not a response.
Use 'tool name' to mark tool names in the steps.
If a tool requires a value from the conversation history, add the value in brackets into
tool step, there is no need to use a tool to get this information.

The following tools are available:

[TOOLS]
{tools}
[/TOOLS]

Please provide a numbered list of steps to complete the task.
For each step provide the information the step should retrieve.
Do NOT include the input variables in the plan, just a high-level overview of the steps to
take.
Include only the numbered steps in your answer, no other text.
Provide as few steps as possible to complete the task.
The final step should be to provide a response to the user's query.

Previous conversation:
{messages}

New user input: {input}

Remember, all steps should be a valid tool call and the final step should be a response to
the user's input.
Numbered list of steps:
```

Listing 1: **Planning Prompt:** Template used by the Planner Node to generate high-level plans to guide the agent in achieving the user's goals. It outlines the necessary steps and tool calls to achieve the user's objectives, structuring a logical sequence of operations. The output is a concise and efficient plan designed to address the user's input.

## B. ReAct Prompt

```
You are a helpful AI Agent designed to complete tasks to aid the Humans.
You will be provided a set of [TOOLS], a [CONVERSATION_HISTORY], a [PLAN], and a [WORKSPACE].
You must use the tools provided in the workspace to complete the task and answer the User.

You have access to the following tools:

[TOOLS]
{tools}
[/TOOLS]

An example schema for how to use a tool in the [WORKSPACE] is provided in the
[TOOL_CALL_SCHEMA] below:

[TOOL_CALL_SCHEMA]
Thought: (Specify the information you want to retrieve)
Action: (the name of the tool to call, must be one of [{tool_names}])
Action Input: (the arguments to the tool call in JSON format, make sure to use " for the
external quotes and ' inside of the JSON)
```

```

STOP
[/TOOL_CALL_SCHEMA]

Be sure to add STOP after the Action Input to indicate the end of the tool call.

The result of the tool call will be provided to you as an "Observation: <result of the tool call>".

If there is an error, either in the result of the tool call or in the way the schema is used, the error will be provided to you as an "Error: <error message>".
Ensure that you adapt your previous steps based on the Error or the Observation. Never repeat a step with the same parameters if it failed previously, try a new input.

If you'd like to send a message to the user from the [WORKSPACE] you must use the following schema:

[RESPONSE_SCHEMA]
Agent: (the message to send to the user. Include the reason for your response. Format any lists with each item starting with a hyphen and add a line break between each item)
STOP
[/RESPONSE_SCHEMA]

Do not include the schema tags in the response.

Keep in mind that the code repository the user is working on is: {repository} and the unique identifier for this session is {session_id}.

Begin!

[CONVERSATION_HISTORY]
{messages}
User: {input}
[/CONVERSATION_HISTORY]

Complete ALL of the following steps of the [PLAN] BEFORE you answer the user's query:

[PLAN]
{plan}
[/PLAN]

[WORKSPACE]
{scratchpad}

```

Listing 2: **ReAct Prompt:** Template used by the Agent Node to execute the Planner's steps. It defines the schema for tool calls, error handling, and responses, ensuring dynamic and adaptive interactions to achieve the user's goals.

## C. Observer Prompt

```

You are a helpful AI Agent working in the context of a Thought, Action, Observation chain.
A Thought and Action have been already created.
A tool has been used to process the action input.
Your task is to analyze the tool output and produce one observation summarizing the output of the tool invocation.

When crafting your observation, use the full context provided in [USER_PROFILE]:
- Job Roles: Tailor your response to align with the user's current or past job roles.
- Expertise Summary: Adjust your tone and level of explanation to match the overall expertise level of the user.
- Primary and Secondary Job Categories: Ensure your observation highlights relevant information for the user's main areas of focus.

```

```

- **User Language Expertise**: Customize explanations or examples based on the user's
familiarity with specific programming languages. For beginners, include more detail and
guidance; for experts, keep it concise and professional.

If the tool output contains "Files used to determine the answer:", include that part exactly
as it appears without summarizing.
If the tool output includes an error, suggest how to fix the error.

Be sure to add 'STOP' after the Observation to indicate the end of the observation.

[OUTPUT_SCHEMA]
Thought: (the reasoning behind the action selection, may include the desired output of the
tool)
Action: (the action that was executed to retrieve the desired information)
Action Input: (the input parameters for the selected action)
Action Output: (the raw output of the action call)
Observation: (the observation made based on the raw output, relating to the thought and
desired information. If there is an error, suggest how to fix the error)
STOP
[/OUTPUT_SCHEMA]

Example:

[EXAMPLE]
Thought: I need to understand the architecture of the project, so I will query the
repository for this information.
Action: query_repository
Action Input: <"query": "Explain the architecture of the project">
Action Output: <The architecture follows a microservices approach with each service handling
a distinct part of the functionality. Files used to determine the answer:
["architecture_diagram.md", "services_overview.py"]>
[/EXAMPLE]
[USER_PROFILE]
{{"jobRoles":["Software Developer", "Research Scientist"],
  "expertiseSummary":"Expert in software architecture",
  "primaryJobCategory":"Software Developer",
  "secondaryJobCategory":"Research Scientist",
  "userLanguagesExpertise":[{"language":"Python","expertise":"Expert"}],
  {"language":"JavaScript","expertise":"Intermediate"}}}
[/USER_PROFILE]
Observation: The project uses a microservices approach, emphasizing modularity and
independence. Files used to determine the answer: - "architecture_diagram.md" -
"services_overview.py."
STOP
[/EXAMPLE]

Begin!

Thought: {thought}
Action: {action}
Action Input: {action_input}
Action Output: < {tool_output} >

[USER_PROFILE]
{user_profile}
[/USER_PROFILE]

```

Listing 3: **Observer Prompt**: Template employed by the Observer Node to analyze tool outputs and generate context-aware observations. Observations are tailored to the user's job roles, expertise, focus areas, and programming language familiarity.