



# A Multi-agent Onboarding Assistant based on Large Language Models, Retrieval Augmented Generation, and Chain-of-Thought

Andrei-Cristian Ionescu  
Delft University of Technology  
Delft, The Netherlands  
a.c.ionescu-1@student.tudelft.nl

Sergey Titov  
JetBrains Research  
Amsterdam, The Netherlands  
sergey.titov@jetbrains.com

Maliheh Izadi  
Delft University of Technology  
Delft, The Netherlands  
m.izadi@tudelft.nl

## Abstract

Effective onboarding in software engineering is crucial but difficult due to the fast-paced evolution of technologies. Traditional methods, like exploration and workshops, are costly, time-consuming, and quickly outdated in large projects. We propose the *Onboarding Buddy* system, which leverages large language models, retrieval augmented generation, and an automated chain-of-thought approach to improve onboarding. It integrates dynamic, context-specific support within the development environment, offering natural language explanations, code insights, and project guidance. Our solution is agent-based and provides customized assistance with minimal human intervention. Our study results among the eight participants show an average helpfulness rating of ( $M=3.26$ ,  $SD=0.86$ ) and ease of onboarding at ( $M=3.0$ ,  $SD=0.96$ ) out of four. While similar to tools like GitHub Copilot, Onboarding Buddy uniquely integrates a chain-of-thought reasoning mechanism with retrieval-augmented generation, tailored specifically for dynamic onboarding contexts. While our initial evaluation is based on eight participants within one project, we will explore larger teams and multiple real-world codebases in the company to demonstrate broader applicability. Overall, Onboarding Buddy holds great potential for enhancing developer productivity and satisfaction. Our tool, source code [7], and demonstration video [6] are publicly available.

## CCS Concepts

- Computing methodologies → Natural language generation;
- Software and its engineering → Programming teams.

## Keywords

Onboarding, Large language models, RAG, CoT, Agents

## ACM Reference Format:

Andrei-Cristian Ionescu, Sergey Titov, and Maliheh Izadi. 2025. A Multi-agent Onboarding Assistant based on Large Language Models, Retrieval Augmented Generation, and Chain-of-Thought. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25)*, June 23–28, 2025, Trondheim, Norway. ACM, New York, NY, USA, 5 pages. <https://doi.org/10.1145/3696630.3728611>



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

FSE Companion '25, Trondheim, Norway

© 2025 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-1276-0/2025/06

<https://doi.org/10.1145/3696630.3728611>

## 1 Introduction

Software engineering is a dynamic domain; the frequency of learning new technologies, frameworks, and tools for software developers is very high. The newcomers also need to adapt to the codebase, project requirements, and team culture rapidly. The domain itself is constantly changing with emerging new trends that may require developers to continuously update their skills [2, 10]. Such challenges may be met through the proper designing of an onboarding system, but such systems are expensive and time-consuming to put into effect for large projects.

Traditional methods, such as documentation and workshops, may no longer be useful, codebase exposure being a more important factor in understanding a project [5]. With recent developments in large language models (LLMs) [1, 8], we can quickly forward relevant advice to each developer [12]. However, managing and implementing this guidance in an effective manner remains a challenge [15].

To address this issue, we introduce the *Onboarding Buddy*. Onboarding Buddy automates the onboarding process by automatically generating detailed, project-specific explanations to users' questions [4]. Doing this cuts down the heavy mentoring needed from other colleagues. The system aims to assist new developers in going faster through the codebase and its logic but also to minimize the frustration and stress associated with onboarding. The system integrates components such as retrieval augmented generation (RAG) [11], and automated chain-of-thought (CoT) [16] with a large language model to present an end-to-end solution, making it more effective for onboarding new team members.

Unlike existing general-purpose LLM coding tools, such as GitHub Copilot, which are general in their assistance, Onboarding Buddy has been designed to tackle the particular challenges of onboarding into new codebases. Its scratchpad mechanism keeps track of the intermediate steps in reasoning and reduces the noise that arises in multi-agent interactions. This is an explicit design choice to offer more explainability and adaptability during onboarding tasks.

To evaluate the Onboarding Buddy, eight users performed onboarding tasks related to setting up their development environment and completing tasks on a backend application. We chose a closed-source project to avoid data contamination with models' opaque training data. Seven out of eight participants successfully completed all tasks, with only one participant making minor coding errors on one of the tasks. After completing the tasks, we asked participants to rate the perceived helpfulness and ease of onboarding on a scale from one to four. As a result, over all the participants and tasks we obtained a high mean rating of perceived helpfulness of ( $M=3.26$ ,  $SD=0.86$ ). Moreover, there was a relatively good mean

ease-of-onboarding rating of ( $M=3.0$ ,  $SD=0.96$ ). Our study has the following contributions:

- Introducing a novel LLM-powered onboarding tool that leverages dynamic and contextualized onboarding experiences.
- Presenting its agent-centric architecture, including specialized agents like the Onboarding Agent,
- Providing the implementation of RAG and automated CoT process, which enhance reasoning and retrieval, and
- Conducting an empirical evaluation of the IntelliJ Onboarding Plugin with eight participants.

## 2 Related Work

Onboarding in software companies involves integrating new developers into the team and project by helping them understand the codebase, architecture, and business logic.

According to Ju et al., [10], effective onboarding is directly related to productivity, job satisfaction, and outcomes like learning and confidence-building. Their case study at Microsoft identified strategies such as the Simple-Complex, Priority-First, and Exploration-Based strategies, which rely heavily on mentoring and peer support to guide new developers. However, mentoring often reduces the productivity of the mentor, placing a significant burden on them. Buchan et al. [3] observed similar trends in Agile teams, where mentoring activities play a key role in successful onboarding.

The current mentor-focused onboarding process, as described by Britto et al. [2], is resource-intensive and less effective in globally distributed projects. Mentoring new developers, especially in remote settings, impacts both mentor and mentee productivity. This paper proposes the Onboarding Buddy as a solution to reduce the mentor burden by providing an autonomous virtual assistant capable of handling routine onboarding queries and supporting the new hires. Onboarding Buddy leverages large language models to offer code explanations, answer project-specific questions, and provide personalized guidance. By automating parts of the onboarding process, this solution aims to increase productivity, reduce stress, and enhance the onboarding experience for both the mentee and the software team the mentee is joining. While tools like GitHub Copilot offer code suggestions, they lack the contextual onboarding capabilities that Onboarding Buddy provides, such as tailored, multi-step guidance and integration with project-specific retrieval tools. Unlike ChatGPT plugins, which focus on broad queries, Onboarding Buddy narrows its scope to optimize onboarding efficiency, i.e., we adhere to the code context in the user's question.

## 3 Onboarding Buddy

Our system takes an agent-centric approach to create the Onboarding Buddy. In this section we will elaborate on the LLM orchestration, prompting techniques, and the technologies we used to create the Onboarding Buddy Q&A capabilities.

In LLM orchestration, a chain is represented by a fixed and rigid sequence of actions. After the chain is implemented, it will act exactly according to the strictly defined steps and without consideration for probable changes in the environment or situation. For instance, when a chain has been developed to process a user question, it will always execute the very same sequence no matter

whether the information is simple or complex, and it cannot adapt if certain data happens to be different from what was anticipated.

Agents behave differently because, in real-time, they would look at the situation with the guidance of an LLM and based on that, decide what and how to perform a certain task. An agent executes an action, then observes the result of that action, and dynamically decides what it is going to do next until the desired result is achieved. That can make the agent much more flexible and responsive. Through these observations and the LLM-guided RAG, Onboarding Buddy mitigates hallucinations or code mismatch errors by cross-checking final answers with retrieved code snippets before presenting them to the user or to the next agent in the chain.

Our LLM onboarding orchestration implies a chain of agents that collaborate with each other to get a final answer as seen in Figure 1. For the foundation of our agent-based system, we utilize OpenAI's GPT-4o [14] model to power the reasoning and response generation capabilities, while OpenAI text-embedding-3-large [13] model handles the embedding functionality in our retrieval system.

- **Contextualization Agent** — Allows the user interaction to be conversational, taking into account past interactions with the system.
- **Onboarding Agent** — Allows the user to ask questions about a software project, and provides tailored answers based on the project and user's needs, generating a chain-of-thought step-by-step plan of action.
- **Step Processor** — A chain-of-thought processor [16] that orchestrates multiple Onboarding Agents in parallel. It accepts the initial step-by-step plan from an **Onboarding Agent** and systematically decomposes each high-level step into more granular sub-steps through a recursive approach. For each sub-step, the processor generates partial solutions, and when a sub-task reaches an appropriate level of refinement, it integrates these partial solutions into a final answer.
- **Message Enhancer Agent** — Allows the system to provide a proper markdown-formatted answer and fix certain inaccuracies in the answer by checking the answer collected from the **Step Processor** against the codebase.

The **Onboarding Agent** is the core of our Onboarding Solution, being the agent that creates the planning and the chain-of-thought used to tackle onboarding tasks.

## 4 Onboarding Agent

This agent has a more complicated structure than the other agents. The 'Onboarding Agent', as seen in Figure 2, is composed of three subcomponents, namely the memory storage, the planning scratchpad, and the retrieval tools. This agent works iteratively and decides dynamically what actions to take to reach an answer using the planning scratchpad and the retrieval tools. For this, we have implemented a custom chain-of-thought approach (CoT) that will be used in the Step Processor.

The approach we developed was inspired by Google's paper that introduced the concept of chain-of-thought prompting [16], where a problem is divided into intermediate reasoning steps, allowing the LLMs to solve more intricate tasks that require a multi-step logic. Instead of getting an immediate answer, the CoT technique forces the model to "reason" into human-like steps. On top of that,

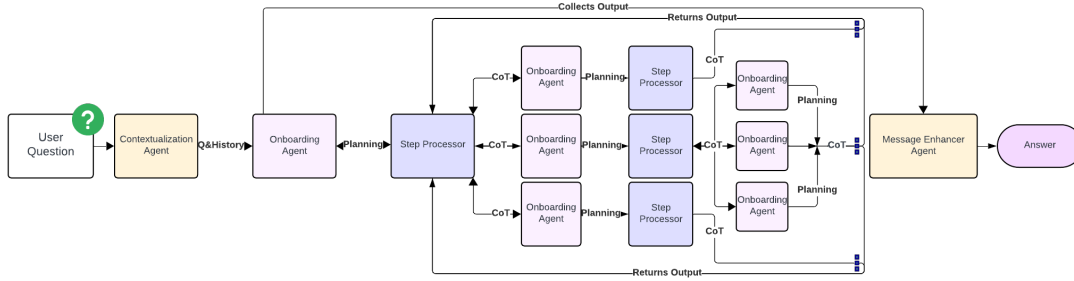


Figure 1: Onboarding Buddy Overall Architecture

Amazon Research [17] elaborated on the same idea from a different angle and developed an automated chain-of-thought generation technique where the manual prompting abilities are minimized. In our approach we used similar ideas, but in an agent-centric approach, where each agent chain in the CoT generation can use external tools such as retrieval tools or code search tools in an exploratory way. For this purpose, each sub-component has its own role in the CoT generation.

#### 4.1 Memory Storage

The memory storage comprises 2 sub-components, a database and a blob storage. The database records the user and LLM’s past interactions, and the blob storage provides information about the context of the question. In our case, the context is based on details about a software project and its documentation. Furthermore, the blob storage allows the retrieval tools to retrieve relevant information based on the planning scratchpad discussed in subsection 4.3.

#### 4.2 Retrieval Tools

The retrieval tools allow the agent to access a software project’s files in the blob storage and perform semantic searches on its contents. With these tools, we built an advanced, LLM-guided, RAG system. The planning scratchpad presented in subsection 4.3) helps the agent build queries and allows the tools to provide feedback on the quality of the query and search results, enabling the agent to adjust its searches or focus on specific search issues.

The semantic search uses FAISS [9], an in-memory dense-vector database that provides efficient similarity search. Each file, including the documentation, is chunked into documents. Each chunked document is represented by data chunks of 2000 characters with

200 characters overlap. Then each chunk is encoded as a vector using an embedding model.

Each chunked document contains as metadata the GitHub URL that locates the source under the main branch and the path in the project, along with a unique numerical identifier per file. Based on this data, we make available two tools to the agent that are “cooperating” to compose an advanced RAG approach and have the following definitions:

- `retrieve_missing_files` - retrieves entire files of source code. The agent can decide to use it only based on the file name information, or if it is guided to do so by the `retrieve_relevant_code_snippets` tool based on the location available in the metadata.
- `retrieve_relevant_code_snippets` - retrieves information chunks via similarity search using a query to the vector database, with two fallback mechanisms: one for empty results with query feedback, and another for repeated queries.

At each iteration, the agent can decide to use any of these two tools, but we define the `retrieve_missing_files` as a more computationally expensive tool since it can blindly retrieve files from the storage. The code snippet retriever returns the top five documents with a similarity score above 0.1, to ensure relevant results. Scores range from 0 (no similarity) to 1 (exact match). The 0.1 threshold filters irrelevant documents but may return zero matches, prompting fallback queries to refine the search. This, in combination with the agent planning scratchpad presented in Section 4.3, allows the agent to self-adjust the queries and files retrieved, and store this information in the agent planning scratchpad.

#### 4.3 Planning Scratchpad

The agent planning scratchpad serves as a dynamic mechanism of internal memory applied by the onboarding agent to track its reasoning process while interacting with different tools, queries, and environments. It works just like a journal or log, where it actually writes down intermediate steps.

When the agent intends to solve any problem, it may invoke a number of tools presented in the section 4.2 or various actions sequentially (e.g., retrieve answers from its own knowledge). Each subsequent action might produce new data or insights, and these would be stored temporarily in the agent planning scratchpad. For instance, if the agent is trying to find some information in the knowledge base (repository or documentation), gather information from the vector store, or reason out something over various facts,

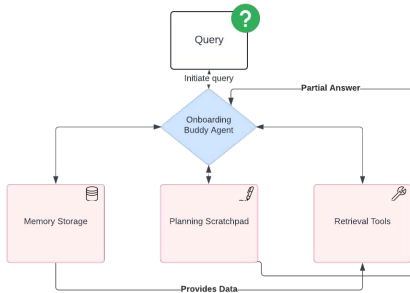


Figure 2: Onboarding Agent Architecture

then at each step it writes it in the scratchpad so that it may refer to it later. This allows the agent to recall why it had chosen to perform some particular action or call some specific tool, making its decision process much more transparent and accurate. Additionally, the scratchpad can help potential users with explainability by providing detailed notes on the CoT process of the agent.

We achieved task chaining using a scratchpad, where each action informs the next to prevent redundancy. For example, an agent can store tool results in the scratchpad for future reference instead of rerunning tools. This also helps guide retrieval tools in the right direction when retrievals fail. If the agent receives new information or needs to change its approach, the scratchpad preserves earlier context while allowing flexibility. Thus, the planning scratchpad serves as a dynamic buffer that considers past decisions and adapts as situations evolve, such as reformulating queries after unsuccessful searches to find the necessary documentation or code snippets.

Moreover, the planning scratchpad is handy during debugging and optimizing agents' workflows. With the contents of the scratchpad, we can identify where an agent might have gone wrong in its reasoning or where an unnecessary call of a tool was made.

In the execution process, the final state of the scratchpad already contains the intermediate results, enabling the agent to assemble a final response that fully answers the user's query while considering the reasoning for each step. The answer of this agent returns a step-by-step plan of action with respect to the user's query. For example, if the user asks how to set up a project, the **Onboarding Agent** provides step-by-step, granular sub-steps of what the user has to perform and how. **Onboarding Agent**'s answer is then used further in the chain to the Step Processor that uses multiple instances of the **Onboarding Agent** in parallel to execute the CoT plan of action created by the initial agent in order to make the steps more precise and granular. Furthermore, the Message Enhancer agent formats, fixes, and beautifies the final answers into a fully readable markdown document.

## 5 Study Setup

To evaluate the usefulness of the Onboarding Buddy, we set up a study. We gathered a sample of eight programmers and assigned each of them three onboarding tasks on an unfamiliar, closed-source project. The first, "*Setup Task*", requested the user to set up the development environment by cloning the repository, uncommenting specific lines of the configuration file, and verifying that everything was correctly set by registering a mock user to the app and testing specific endpoints. The second task, "*New Payment Option Task*", required the implementation of a new payment option in the in-application subscription system from which one had to understand an existing payment pipeline and adjust certain code sections. Finally, the "*Questionnaire Duplication Task*" entailed writing a new API endpoint for duplicating questionnaires by implementing duplication logic and writing integration tests to ensure functionality.

To assess the usefulness of the approach, we collected project telemetry data to track the progress of tasks (e.g., code edits). After completing the tasks, we asked participants to evaluate the helpfulness and ease of onboarding. Both questions required participants to rate the entire system on a scale from 0 to 4, where 0 signifies "not helpful" or "very difficult", and 4 indicates "very helpful" or

"very easy", respectively. Lastly, we requested open-ended feedback, inviting specific user suggestions for tool enhancements.

Our pilot study involved eight participants on one codebase due to the limitations of the demonstration project. This may affect generalizability. In future evaluations at JetBrains, we aim to include multiple teams with different technology stacks and compare Onboarding Buddy's performance to off-the-shelf solutions like GitHub Copilot and ChatGPT plugins on similar tasks.

## 6 Results

Seven out of eight participants successfully completed all tasks, with only one participant encountering minor coding errors on a single task. Questionnaire results indicated that the perceived helpfulness of the Onboarding Buddy for all tasks had a high mean of ( $M=3.26$ ,  $SD=0.86$ ). Moreover, the ease-of-onboarding ratings averaged ( $M=3.0$ ,  $SD=0.96$ ), reflecting a smooth onboarding experience. These mean values represent the combined responses of all study participants across all tasks. Task completion accuracy was close to 100% among participants who marked their tasks as finished but also based on the collected telemetry. However, a number of users experienced minor hiccups during their onboarding process, particularly on set-up and questionnaire duplication tasks, which were the hardest tasks we exposed our participants to.

The most common suggestions were about user experience (UX), including the following: more context awareness by providing more specific code snippets along with line numbers, more user-friendly UI features such as multi-line input and some navigation buttons, and faster response times. Although not a formal benchmark, participants who used mainstream AI coding tools in their daily workflow noted in their feedback that Onboarding Buddy gave more context-specific advice, particularly on setup tasks. This suggests potential advantages over generic code assistants, though rigorous A/B testing shall be done.

## 7 Conclusion

Overall, the Onboarding Buddy is an easy-to-use and useful tool that helps users in performing development tasks. This design is modular and adaptive, capable of efficiently solving problems using techniques like chain-of-thought reasoning and retrieval tools. However, addressing the identified shortcomings, enhancing technical reliability, and optimizing the user interface will significantly improve the overall usability and utility of the Onboarding Buddy. Refining these aspects will make the Onboarding Buddy capable of providing a smoother and more effective onboarding experience, thereby further enhancing user satisfaction and productivity. In future work, we will incorporate more robust validation checks (e.g., hallucination mitigation), and benchmark Onboarding Buddy against GitHub Copilot and ChatGPT-based solutions in varied (enterprise) settings. Regarding scalability, future versions will involve live project updates, reducing reliance on pre-encoded project data.

## 8 Acknowledgment

This work was conducted as part of the AI for Software Engineering (AI4SE) collaboration between JetBrains and Delft University of Technology. The authors gratefully acknowledge the financial support provided by JetBrains, which made this research possible.

## References

- [1] Ali Al-Kaswan, Toufique Ahmed, Maliheh Izadi, Anand Ashok Sawant, Premkumar Devanbu, and Arie van Deursen. 2023. Extending source code pre-trained language models to summarise decompiled binaries. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 260–271.
- [2] Ricardo Britto, Daniela S. Cruzes, Darja Smite, and Aivars Sablis. 2017. Onboarding software developers and teams in three globally distributed legacy projects: A multicase study. *Journal of Software: Evolution and Process* 30, 4 (11 2017).
- [3] Jim Buchan, Stephen MacDonell, and Jennifer Yang. 2019. Effective team onboarding in Agile software development: techniques and goals. arXiv:1907.10206 [cs.SE] <https://arxiv.org/abs/1907.10206>
- [4] Amirhossein Deljouyi, Roham Koohestani, Maliheh Izadi, and Andy Zaidman. 2024. Leveraging large language models for enhancing the understandability of generated unit tests. *arXiv preprint arXiv:2408.11710* (2024).
- [5] Neil A. Ernst and Martin P. Robillard. 2023. A study of documentation for software architecture. *Empirical Software Engineering* 28, 122 (September 2023), 1–32. doi:10.1007/s10664-023-10347-2 Accepted 25 May 2023, Published 15 September 2023.
- [6] Andrei-Cristian Ionescu, Sergey Titov, and Maliheh Izadi. 2025. Onboarding Buddy - Demonstration Video. <https://youtu.be/V18mjpziCvo>. Accessed: 2025-03-27.
- [7] Andrei-Cristian Ionescu, Sergey Titov, and Maliheh Izadi. 2025. Onboarding Buddy - Tool and Source Code. <https://onboarding.software/register?colab=true>. Accessed: 2025-03-27.
- [8] Maliheh Izadi, Jonathan Katzy, Tim Van Dam, Marc Otten, Razvan Mihai Popescu, and Arie Van Deursen. 2024. Language models for code completion: A practical evaluation. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.
- [9] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. <https://github.com/facebookresearch/faiss>. Accessed: 2024-09-27.
- [10] An Ju, Hitesh Sajjani, Scot Kelly, and Kim Herzig. 2021. A Case Study of Onboarding in Software Teams: Tasks and Strategies. arXiv:2103.05055 [cs.SE]
- [11] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:2005.11401 [cs.CL] <https://arxiv.org/abs/2005.11401>
- [12] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. 2023. Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. ACM.
- [13] OpenAI. 2023. text-embedding-3-large. <https://platform.openai.com/docs/guides/embeddings>. Language model.
- [14] OpenAI. 2024. gpt-4o. <https://platform.openai.com/docs/models/gpt-4o>. Language model.
- [15] Agnia Sergeyuk, Ekaterina Koshchenko, Ilya Zakharov, Timofey Bryksin, and Maliheh Izadi. 2024. The Design Space of in-IDE Human-AI Experience. *arXiv preprint arXiv:2410.08676* (2024).
- [16] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903 [cs.CL] <https://arxiv.org/abs/2201.11903>
- [17] Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2022. Automatic Chain of Thought Prompting in Large Language Models. arXiv:2210.03493 [cs.CL] <https://arxiv.org/abs/2210.03493>