

Developer Experiences with a Contextualized AI Coding Assistant: Usability, Expectations, and Outcomes

Gustavo Pinto
Zup Innovation & UFPA
Belém, PA, Brazil
gustavo.pinto@zup.com.br

Igor Steinmacher
Northern Arizona University
FlagStaff, AZ, EUA
Igor.Steinmacher@nau.edu

Cleidson de Souza
UFPA
Belém, PA, Brazil
cleidson.desouza@acm.org

Alberto de Souza
Zup Innovation
São Paulo, SP, Brazil
alberto.tavares@zup.com.br

Thayssa Rocha
Zup Innovation & UFPA
Belém, PA, Brazil
thayssa.rocha@zup.com.br

Edward Monteiro
StackSpot
São Paulo, SP, Brazil
edward.monteiro@stackspot.com

ABSTRACT

In the rapidly advancing field of artificial intelligence, software development has emerged as a key area of innovation. Despite the plethora of *general-purpose* AI assistants available, their effectiveness diminishes in complex, domain-specific scenarios. Noting this limitation, both the academic community and industry players are relying on *contextualized* coding AI assistants. These assistants surpass general-purpose AI tools by integrating proprietary, domain-specific knowledge, offering precise and relevant solutions. Our study focuses on the initial experiences of 62 participants who used a contextualized coding AI assistant — named StackSpot AI— in a controlled setting. According to the participants, the assistants’ use resulted in significant time savings, easier access to documentation, and the generation of accurate codes for internal APIs. However, challenges associated with the knowledge sources necessary to make the coding assistant access more contextual information as well as variable responses and limitations in handling complex codes were observed. The study’s findings, detailing both the benefits and challenges of contextualized AI assistants, underscore their potential to revolutionize software development practices, while also highlighting areas for further refinement.

KEYWORDS

LLM, LLM-based applications, User expectations, Perception of Productivity

ACM Reference Format:

Gustavo Pinto, Cleidson de Souza, Thayssa Rocha, Igor Steinmacher, Alberto de Souza, and Edward Monteiro. 2024. Developer Experiences with a Contextualized AI Coding Assistant: Usability, Expectations, and Outcomes. In *Conference on AI Engineering Software Engineering for AI (CAIN 2024)*, April 14–15, 2024, Lisbon, Portugal. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3644815.3644949>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CAIN 2024, April 14–15, 2024, Lisbon, Portugal

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0591-5/24/04...\$15.00

<https://doi.org/10.1145/3644815.3644949>

1 INTRODUCTION

In the rapidly evolving landscape of artificial intelligence (AI) and its integration into various industries, the software development domain remains at the forefront of innovation [8]. Developers today are equipped with an unprecedented set of coding tools and AI agents, designed to navigate and simplify the complexities of software engineering. However, as software projects grow intricate, the demand for precise and efficient coding assistance becomes critical [7].

Despite their groundbreaking nature, general-purpose AI assistants currently have a fundamental shortcoming: they often provide generic or inaccurate responses, particularly when confronted with contextualized, domain-specific queries [5]. This gap is felt by developers who seek guidance, for example, in tasks related to optimizing a database query [1] or deciphering the complexities of a proprietary codebase. Conventional AI tools, while advanced in many aspects, often fall short of delivering the depth and specificity required in these scenarios. Such limitations not only hinder productivity but also pose a barrier to harnessing the full potential of AI in software development.

The industry is responding to these challenges by developing contextualized coding AI assistants. These tools, underpinned by advanced AI models, are specifically designed to access and utilize proprietary, domain-specific knowledge, which general-purpose assistants typically lack. This specialized approach enables them to offer targeted assistance, especially useful in complex, domain-specific scenarios. To illustrate, imagine a scenario where a developer is working on an intricate e-commerce platform, and they encounter a challenge related to optimizing a multi-tier product recommendation algorithm. While a general-purpose AI might offer broad-based guidance or algorithmic solutions, a contextualized coding assistant, familiar with the proprietary nuances of that specific e-commerce platform and its surroundings, might pinpoint exact issues based on historical data or even provide solutions that account for platform-specific constraints, or company-based frameworks. In essence, while a general-purpose AI tool might suggest generic best practices, a contextualized assistant could reference company-specific requirements documents and related projects, offering answers that are not only effective but also tailored to that organization’s unique needs. For instance, Enterprise Tabnine

customers might choose to train their own custom model based on their company’s source code¹.

These contextualized tools, while differing in purpose from general-purpose coding assistants, often share a common technological foundation: they leverage the Retrieval-Augmented Generation (RAG) technique [9]. This involves retrieving relevant information from specialized sources and generating contextualized responses using advanced language models. An example of this is the contextualized coding AI assistant name StackSpot AI developed by ZUP INNOVATION²— a software partner tech company — aimed at enhancing developer productivity, confidence, and experience with AI-based tools. The main capabilities of this assistant are detailed in Section 3.

In this paper, we report the findings from a study about the experience of 62 practitioners who used the StackSpot AI assistant for the first time in a controlled, online environment. During four hours, they were introduced to concepts and usage details of StackSpot AI, followed by a hands-on experience performing simple tasks. The participants interacted and provided feedback during the whole online discussion. The analysis of their feedback highlights several key [B]enefits and [C]hallenges encountered:

- [B] Generation of accurate codes for swift integration with internal APIs and support for routine tasks;
- [B] Time efficiency by centralizing information access;
- [B] Streamlined access to documentation and guidelines within the IDE;
- [C] Multiple knowledge sources are required to maintain response accuracy;
- [C] Inconsistency in responses to identical prompts, requiring prompt refinement; and
- [C] Difficulties in generating complex code structures.

Furthermore, the participants provided important insights, bringing feedback on the experience, suggestions on new functionalities, and reflections on the productivity and reliability of code generated by StackSpot AI.

2 RELATED WORK

Research in AI assistants has focused on different aspects including the benchmarks necessary to evaluate and compare them [6], the correctness [19], complexity [13], quality [12], and security [15] of the generated code, the developers’ experience while using these assistants [3, 17], among other aspects. In this paper, we are interested in two aspects. First, the user experience using these tools. Second, the correctness of the solutions generated, i.e., their ability to, given a particular problem, generate a code solution that will actually solve that problem. This is measured by checking whether the solution passes the test cases associated with the original problem.

2.1 User Experience

We can find a few papers discussing the user experience of software developers using AI code assistants including [1, 3, 4, 17?, 18]. In general, these studies indicate that developers save time using AI assistants, i.e., “interactions with programming assistants are

bimodal: in acceleration mode, the programmer knows what to do next and uses Copilot to get there faster; in exploration mode, the programmer is unsure how to proceed and uses Copilot to explore their options” [3]. Even when the assistants are not 100% correct, they still generate code that can be used as a “starting point” for further work.

These studies also reported some of the limitations of these tools, mainly lack of correctness of the code suggestions and interruptions, i.e., the assistants disturb the natural flow of work [1, 4]. More interestingly, they report coping strategies to deal with Copilot’s limitations: “to accept the incorrect suggestion and attempt to repair it,” add more context so that the assistant improves its suggestions, or simply stop using the tool.

2.2 Correctness

In 2022, two different papers were published assessing the correctness of GitHub Copilot. In the first paper, Nguyen and Nadi [13] assessed the correctness of Copilot’s suggestions in four different programming languages: Java, JavaScript, Python, and C. Each programming language had a different result with Python code generated by Copilot with a 42% correctness, while Java had 57% and JavaScript with 27%. These authors tested the code generation abilities to solve 33 questions randomly selected from LeetCode, a popular Question Pool website with several various coding questions on different topics (array, algorithm, sorting, etc).

Meanwhile, Yetistiren and colleagues [19] focused solely on Python and used the HumanEval [6] benchmark, the same one used to evaluate Codex, the GPT model behind Copilot. This benchmark contains 164 *original* programming problems “with some comparable to simple software interview questions”. In their result, Copilot’s suggestions had a 28.7% correctness rate.

Several factors might explain the different correctness rates in these studies (42% vs 27.8%). Arguably, a potential explanation is associated with the datasets used. While Nguyen and Nadi [13] used a *popular* programming site, Yetistiren et al. [19] used *original* programming problems, i.e., a popular programming site like LeetCode might even be used in the Copilot’s training dataset. This seems to suggest that Copilot’s correctness is influenced by the presence of similar data in its training dataset. Therefore, when faced with domain-specific queries, Copilot is likely to provide generic or inaccurate suggestions.

3 STACKSPOT AI

In this section, we describe how StackSpot AI works.

3.1 Approach

Different than Copilot or CodeWhisperer, which are *general-purpose* coding assistants, StackSpot AI is a highly *contextualized* coding AI assistant. StackSpot AI takes into account the nuanced requirements of individual developers and the intricacies of specific projects (codebases). This tailored approach is based on the implementation of the Retrieval Augmented Generation (RAG) mechanism [9].

RAG is an approach designed to enhance LLM-generated content by anchoring it in external knowledge sources. In question-answering systems, RAG accesses up-to-date, reliable information

¹<https://www.tabnine.com/code-privacy>

²StackSpot AI and ZUP INNOVATION are two pseudonyms adopted for double anonymous purposes.

and provides transparency to users regarding the model’s information sources, promoting trust and verifiability. So, this approach mitigates the risk of sensitive data leakage and misinformation generation while also improving response quality. An illustrative list of possible knowledge sources includes:

- (1) An extensive catalog of APIs recurrently harnessed by the development team;
- (2) Exemplary code snippets serving for discerning coding paradigms or facilitating code modernization activities;
- (3) Customized artifacts written in natural language, including but not limited to, guidelines delineating the protocol for repository commits and a comprehensive list of software requirements to be implemented.

By providing relevant and up-to-date information to the LLM, RAG also reduces the need for constant model retraining and parameter updates, lowering computational and financial overhead [9], since there is no need to build a new foundation model or retrain an existing one. Additionally, RAG is a two-pronged structure consisting of “retrieval” and “generation” components [9].

The “retrieval” facet of RAG is designed to fetch relevant documents from a specified dataset. Traditional databases might falter in efficiently retrieving relevant documents. In StackSpot AI, we use information retrieval techniques to identify the most relevant document for a given user query. Although the retrieval component is efficient at sourcing relevant information, it does not have the capability to generate new content.

On the flip side, the “generation” component harnesses the prowess of OpenAI’s most recent model, GPT-4. Imagine a scenario where a developer is conceptualizing a new algorithm but hits a roadblock in terms of its implementation. StackSpot AI, channeling the generative capabilities of GPT-4, can aid in generating code snippets that are tailored to the developer’s specific context, based on the documents found by the retrieval component.

In essence, StackSpot AI joins advanced contextual retrieval with state-of-the-art generative capabilities, ensuring developers receive precise, contextual, and timely assistance. It does so by using OpenAI’s newest model, GPT-4.

3.2 Prep-and-Go

StackSpot AI has two main interfaces. The first one is a web portal in which users can configure their teams’ preferences and upload representative documents, which would be later used by the retrieval component. These preferences’ configurations allow the use of recommended development tech stacks and code patterns that are often employed in the development team. As such, the generated code might respect these stacks, minimizing the developer’s effort in translating the generated response into their codebase.

Once the configuration is done, users can turn their attention to the StackSpot AI plugin that is currently available for VSCode and IntelliJ. Using this plugin, users could interact with the second interface: a coding assistant chatbot. In this way, developers can craft prompts, refine the answers, and copy the generated solution to the code editor. Additionally, it’s important to note that StackSpot AI, utilizing GPT-4 as its core LLM, needs to manage the token limit imposed per request effectively. This token limit is crucial because exceeding it can lead to prompt overflow, a scenario where

the number of tokens used exceeds the LLM’s capacity. Given that StackSpot AI functions as a chatbot, it tracks and retains a history of the most recent messages exchanged with users. This historical data enriches the input prompt, enhancing the context and relevance of StackSpot AI’s responses. However, to prevent prompt overflow and maintain efficiency, StackSpot AI implements a strategy of selectively discarding older messages. This process ensures that the prompt remains within the token limit while retaining the most pertinent and recent interactions. Additionally, recognizing that users may shift topics during a conversation, StackSpot AI is designed to dynamically adjust which messages it retains. It prioritizes those that are most relevant to the current context of the dialogue. This adaptive approach ensures that StackSpot AI remains focused and relevant to the user’s immediate needs, despite the evolving nature of the conversation.

Figure 1 shows an example of the use of the StackSpot AI plugin on VSCode. As one can see, StackSpot AI combines a code editor with a chat interface. The red box indicates the knowledge source found in the user search.

Finally, as a conversational agent, StackSpot AI extends its capabilities beyond mere generation of code snippets. It can engage users in broader discussions on various programming topics, offering insights and clarifications [16]. Additionally, it plays a role in enhancing users’ programming skills through interactive learning and guidance, providing a more comprehensive, educational experience in the realm of software development.

4 USER STUDY

To evaluate the developers’ experience in using StackSpot AI, we organized an in-company online study. The study was performed within ZUP INNOVATION, a large software-producing company, with around 3.5k employees, and more than 10+ years in the market, working with some of the largest financial institutions in Latin America and abroad. For instance, for one of their clients, ZUP INNOVATION engineers rewrote millions of Cobol legacy code into modern programming languages, helping to move their physical infrastructure to the cloud. In such a context, there is an important need for a text-based coding assistant, in particular, for modernization tasks.

The goal of the study was to introduce practitioners to StackSpot AI and gather representative feedback to improve the product’s quality and usability. We intentionally refrained from setting specific design objectives for the tool, tailoring it to particular user groups (such as novices or experts) or specific scenarios (like coding or learning new programming languages), as we wanted any value provided by StackSpot AI to emerge from our user study.

In this section we describe how the study was planned, (Section 4.1), how we collected data (Section 4.2), and how we analyzed the data collected (Section 4.3).

4.1 Study Planning

The study was structured to provide participants with a journey into the capabilities and functionalities of StackSpot AI. Through an online meeting, facilitators stepped from introducing concepts and product demonstrations to hands-on exploration and collaborative discussions. The intention was to provide a comprehensive

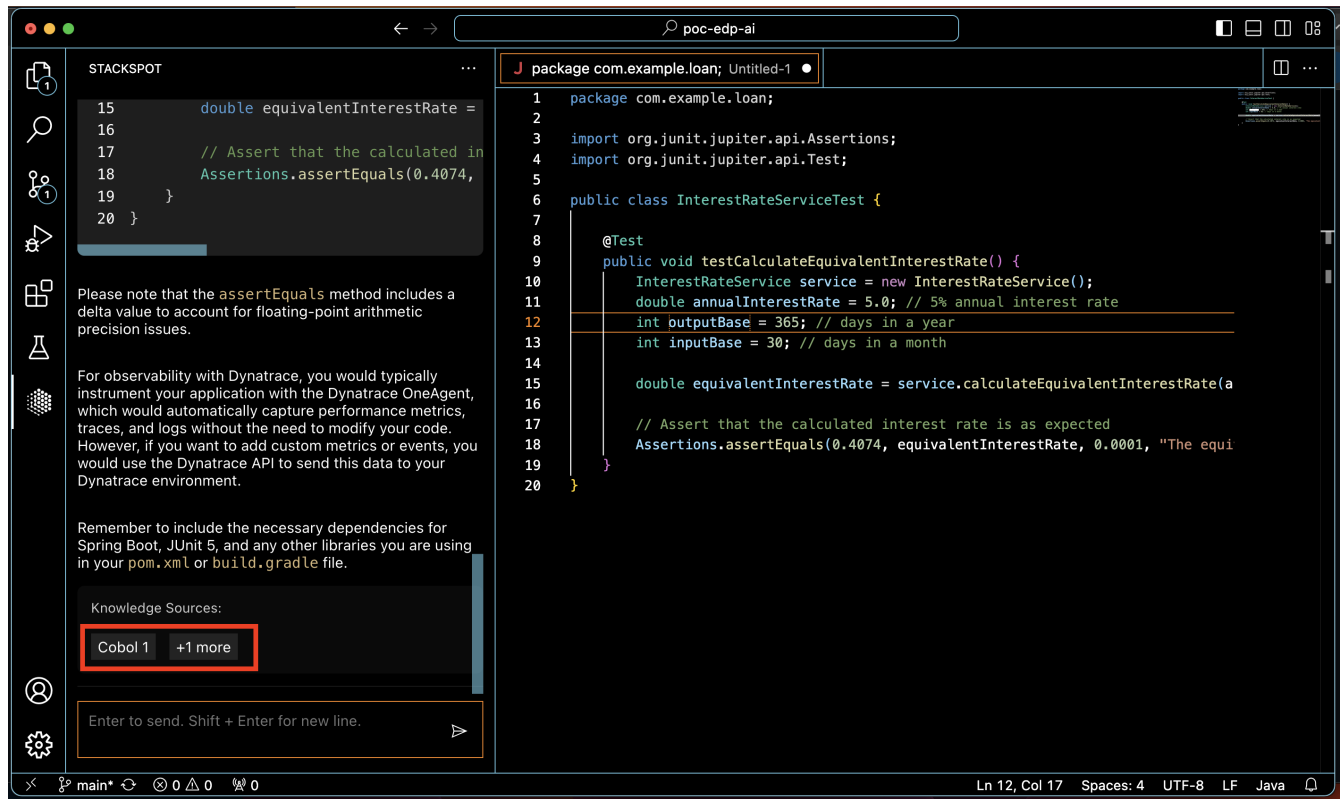


Figure 1: StackSpot AI plugin on VS Code.

understanding of StackSpot AI. To do so, the planned timeline for the meeting was:

- **Welcoming Session (2:00 PM – 2:15 PM):** An initial moment to introduce ourselves and encourage those who did not answer the pre-study questionnaire to fill it out (more details about this survey are available in Section 4.2).
- **StackSpot AI Demonstration (2:15 PM – 2:50 PM):** We conducted an StackSpot AI demonstration, emphasizing its key features and ways of interaction. By the conclusion of the demonstration, participants would be equipped to execute a basic “hello world” activity using the platform.
- **Exploring StackSpot AI (3:00 PM – 4:50 PM):** With the environment duly set up, participants were engaged in independent exploration. For this exercise, we provided three curated knowledge sources and specific tasks to facilitate hands-on experience. These tasks encompassed various activities: 1) replicating the initial demonstration, 2) employing the given knowledge sources to simulate banking operations like transfers or payments, 3) refining the user’s prompt for more effective use of the knowledge sources, and 4) exploring extra knowledge sources relevant to the participant’s specific field.
- **Group Discussion (5:00 PM – 6:00 PM):** In this concluding hour, we sought to address and reflect upon their experience while utilizing StackSpot AI. We made it clear that the developers’ insights and feedback were invaluable in the ongoing refinement of the StackSpot AI.

A discerning reader might observe that there is a 10-minute break between each session. This interlude was intentionally scheduled to offer participants a chance for stretching, restroom breaks, and other necessities. Moreover, to build rapport and foster engagement with attendees who remained in the room, we initiated discussions about the preceding activity, inquiring about any uncertainties or challenges they may have encountered, for instance, in their machine configuration.

4.2 Data Collection

We collected data in three different moments. First, we administered a pre-study survey (Section 4.2.1); second, we audio-recorded the conversations that happened during the study (Section 4.2.2); and, third, we conducted a group discussion after the study (Section 4.2.3). We will discuss each of these collection methods next.

4.2.1 Survey. We established the survey as an online questionnaire. Before the official survey release, we piloted the instrument with three practitioners to assess its clarity and relevance. The feedback from these pilot participants allowed us to refine certain queries, ensuring optimal comprehension — for instance, we adjusted a question about user experience to be more specific based on a suggestion. Following these revisions, the pilot responses were purged to ensure the integrity of the final dataset.

The survey was disseminated company-wide, via our weekly news email, one week before the study. To maximize engagement,

the survey was also promoted in various company communication channels. As a pre-requisite to join the session, employees were required to complete the questionnaire. Additionally, at the outset of the study, we emphasized the importance of the survey, allotting the initial 15 minutes for attendees to fill it out. Ultimately, a total of 62 practitioners participated, though pinpointing an exact response rate proved challenging, given the broad outreach juxtaposed with the targeted audience for the study. The survey was crafted using the TypeForm platform; the platform estimated an average completion time of four minutes.

Questions. Our survey had 12 questions (all of them were required and five were open). The survey was not anonymous; in the very first question, we asked for the participant's email. We did so to generate the invite list to the online room, the second phase of the data collection. The questions covered in the survey were:

- Q1) Enter your email? [Open question]
- Q2) What is your age? [Open question]
- Q3) What is your technical profile? [Open question]
- Q4) How long have you been working with this technical profile? [Numerical scale {0 to 10}]
- Q5) How would you rate your experience in the following programming languages? [Numerical scale {1 to 5}], for the following programming languages: Java, C#, Go, Python, JavaScript, and TypeScript
- Q6) Have you ever used a Generative AI tool for code generation? [Choices: {Yes/No}]
- Q7) Which Generative AI tool for code generation do you use most frequently? [Multiple Answer: {Github Copilot, Amazon Whisperer, Sourcegraph Cody, Other}]
- Q8) How often do you use this tool? [Numerical scale {1 to 5}]
- Q9) How useful is the output from these tools to you? [Numerical scale {1 to 5}]
- Q10) Do you need to modify the code generated by these tools before making a commit? [Numerical scale {1 to 5}]
- Q11) What features provided by these tools do you find most interesting? [Open question]
- Q12) What features would you like these tools to implement? [Open question]

The complete set of questions, as well as the actual survey responses, are anonymized and available at the companion website³.

4.2.2 Recorded conversations. All discussions and interactions that took place during the study were recorded, having obtained the explicit consent of the participants. At the welcoming session, we clarified that attendees who might be hesitant about the recording could still actively engage with the tool. However, we requested that they refrain from joining the public discussions through video or text. Instead, they were encouraged to communicate using specified private channels. Notably, we found that every attendee was receptive to the recording procedure, with none opting for the private communication channels. The total duration of the recorded video amounted to 4 hours and 2 minutes.

4.2.3 Group discussion. Our group discussion mirrored the one adopted by Luz and colleagues [10]. The organization of the group

discussion was as follows: (1) a researcher-moderator helmed the session, outlining discussion subjects for the participants; (2) as each topic was broached, participants presented their thoughts, and keywords were posted on the shared slides; (3) subsequently, with the notes on the slides, the participants could provide additional comments.

The group discussion happened immediately after the technical session. We used the same Google Meet call to conduct the discussion. However, many practitioners were unable to attend the whole study due to other commitments. Therefore, the discussion started with 34 participants and ended with 20. When we inspected the recorded video, we observed several interactions among the participants, usually complimenting StackSpot AI. So, although the group discussion concluded with 20 participants, we actually had around 25 participants engaged in the conversation (not to mention the interactions via chat in the meeting room).

Although we tried to reach different participants in the online room, due to the high number of attendees, not all of them were able to express their perspectives. The group discussion lasted approximately 1 hour and we sought to answer the following questions during this phase:

- Did you find any issues with running StackSpot AI that halted your progress?
- Did you feel the need to understand more about the provided knowledge sources?
- How did you perceive the ease of use of StackSpot AI? (What factors influenced this evaluation?)
- What were the primary benefits you derived from using StackSpot AI in your project?
- What challenges did you face when using StackSpot AI?
- How useful and accurate were the responses generated by StackSpot AI for your purpose?
- What other features would you expect StackSpot AI to offer?
- Did StackSpot AI save you time during the development process? (If yes, how?)
- How likely you would be to integrate StackSpot AI into your daily work routine?

4.3 Data Analysis

We employed diverse data analysis methods, according to the data collected. To analyze the survey delineated in Section 4.2.1, we used descriptive statistics to provide a concise summary of the primary information. For the open-ended questions, open coding techniques were utilized to classify the answers.

To analyze the recorded conversations and the group discussion (Section 4.2.2 and Section 4.2.3) we made use of a distinct approach. We developed a software tool that automatically downloaded the video, extracted its audio, and leveraged the OpenAI Whisper model⁴ for transcription. This process yielded text data comprising 29,851 words (45,580 tokens⁵). To identify predominant categories and themes within this text, we queried GPT-4. For example, GPT-4 assisted in enumerating the most recurrent questions posed during the meeting and pinpointing prevalent issues highlighted by participants. We then manually refined GPT-4's output,

³To be published upon acceptance.

⁴<https://github.com/openai/whisper>

⁵Computed with Tiktoken library, <https://github.com/openai/tiktoken>.

	1	2	3	4	5
Java	18.9%	18.4%	12.6%	25.7%	24.3%
C#	59.3%	21.1%	7.2%	5.7%	6.7%
Go	80.6%	11.1%	6.7%	1.7%	0%
Python	40.5%	27.2%	21%	8.2%	3.1%
JavaScript	23.4%	24.4%	25.9%	17.4%	9%
TypeScript	40.1%	19.3%	19.3%	12.7%	8.6%

Figure 2: Participant Programming Experience.

supplementing it with pertinent observations that the model might have overlooked. We conducted two approaches as a way to mitigate study hallucination problems⁶. First, one author read the full transcript while watching the recorded video; during this task, the author fixed minor errors in the transcript, making it more accurate. Second, we asked two practitioners who joined in the study to analyze the list of categories and themes produced by GPT-4, remove them if they found them wrong, and complement them with additional ones that they found representative (although missing from the initial list). The practitioners mentioned that the categories in the GPT-4 list are accurate and no additional items were provided.

The researchers involved also analyzed the themes and the data, and, although they agreed with GPT-4 classification, they judged that there was some overlap across the categories. After discussions, they came to a consensus on keeping four main categories: 1) general questions, 2) perceived benefits, 3) challenges encountered, and 4) perception of productivity. GPT-4 suggested one other category, called “usefulness of the generated answer”. We dismissed this last category for the sake of traceability since their themes were following up on those themes from categories 2, 3, and 4. We elaborate on each one of these categories throughout Section 5.

4.4 Participants Demographics

In this section, we present the demographic details of the 62 respondents to our survey.

The average age of our participants is 34 years, with 6 years of experience in software development, and 2.6 years affiliated with the company. When evaluating proficiency in programming languages (Figure 2), a significant portion (50%) self-identified as proficient in Java (columns 4 and 5 in Figure 2). This was followed by JavaScript (26.4%), TypeScript (21.2%), C# (12.4%), Python (11.2%), and Go (1.7%). Figure 3 shows the percentage of the participants who had experience with GenAI tools (Figure 3.a), and which ones (Figure 3.b). Notably, 83.9% of the respondents have prior experience with coding AI assistants, with GitHub Copilot being the predominant choice (57.7%).

⁶Hallucinations are common in LLM-based tools. See Bang’s et al. discussion of ChatGPT [2].

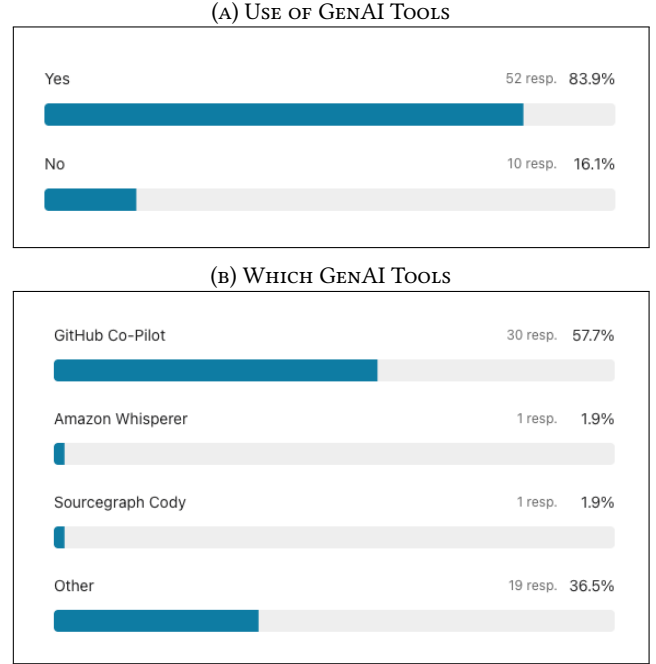


Figure 3: Participants experience with GenAI-based tools.

Other AI coding assistants, each mentioned by a single respondent, include Amazon Code Whisperer, Sourcegraph Cody, AskCodi, Codeium, Phind, and ChatGPT (with 11 mentions). A noteworthy observation is that 58.8% of the participants seldom utilize these tools; in fact, 17.6% have never employed them. A minority, 23.5%, incorporate these tools regularly in their workflow. When assessing the utility of the output from these assistants, 29% of the participants found them useful, and an additional 19.4% deemed them highly valuable. However, 46.7% reported that they often make substantial modifications to the outputs these tools produce. The rest, 4.9% had a neutral answer to this question.

5 RESULTS

Given the exploratory nature of our study, we organize our results according to the four themes that emerged in our analysis.

5.1 General Questions

We start by describing general questions that the participant had when first interacting with StackSpot AI. Understanding general questions is important because developers may need to answer them before they can write any code.

What are the main features and functionalities of StackSpot AI? Participants expressed interest in understanding how StackSpot AI operates and its distinction from other generative AI tools in terms of feature set. They sought detailed insights into the specific functionalities that set StackSpot AI apart, including its unique capabilities and how these functionalities enhance the user experience compared to other available tools.

How to set up and start using StackSpot AI? This question focused on the setup and initial configuration requirements for StackSpot AI. Participants, who were used to other GenAI tools with minimal setup, inquired about a similarly streamlined process for StackSpot AI. They were particularly interested in StackSpot AI's integration with various IDEs via plugins, concerning support for a wide array of IDEs not yet compatible (such as XCode and Eclipse). Additionally, there was a keen interest in understanding the role and selection of knowledge sources for StackSpot AI, including the number and types of sources to use for optimal performance.

What are the benefits of using StackSpot AI compared to other similar tools? Participants, experienced with other GenAI tools, frequently asked about StackSpot AI's advantages over these tools. A notable inquiry centered on the efficacy of StackSpot AI in the absence of user-provided knowledge sources, especially in comparison to tools like CoPilot, which also utilize state-of-the-art OpenAI models. We clarified that without specific knowledge sources, StackSpot AI may not demonstrate significant performance enhancements. This discussion underscored that StackSpot AI is not a *panacea*; effective use requires users to actively engage in the selection and design of knowledge sources to fully leverage its capabilities.

Does StackSpot AI actually bring productivity gains and time savings for developers? Another recurrent general question was about the tangible benefits of StackSpot AI in enhancing software development productivity⁷. Participants were keen to learn how StackSpot AI translates its features and capabilities into real-world time savings and efficiency improvements for developers. One respondent said that *"I asked him [the tool] to generate unit tests, he did, the tests were good. I said, that's good, but I want another one. He went and did it right too. I saw that he really can generate test and now it's something I'll have to worry less, because I can leave it up to him to do it. I think this will help a lot on a daily basis. It'll speed things up a lot."*

5.2 Perceived Benefits

In this section, we delve into the potential contributions of StackSpot AI in simplifying and enhancing the coding process. The participants' experiences, initially tinged with skepticism, evolved into recognition of StackSpot AI's capabilities in generating precise code for complex API integrations. We explore how StackSpot AI's contextual understanding, quick commands, and interactive code refinement through chat significantly streamline the development cycle.

Generation of accurate codes for swift integration with internal APIs Initially, participants expressed skepticism about the accuracy and completeness of the code generated by StackSpot AI, particularly for complex API integration. During the study, the participants were handed a few API files, which they used to generate integration using StackSpot AI. Their experience revealed StackSpot AI's ability to generate functional code snippets, acknowledging this as a crucial feature for accelerating development cycles and reducing errors. This was highlighted by P4, who mentioned the

following: *"Thus, the tool accurately grasped the context, generating a code with Spring framework — it was impressive. I did not expect it to work out so well. StackSpot AI created the class as I requested. Indeed, I am quite amazed here."*

Faster and more context-aware responses and code suggestions compared to other tools. Participants initially perceived StackSpot AI as an enhancement to general-purpose AI coding assistants, intrigued by its potential for quick, context-sensitive solutions. Upon further use, they recognized StackSpot AI's significant value in providing time-efficient, relevant, and precise coding suggestions specifically tailored to their project's context. As participant P5 mentioned: *"For me, the contextualization helped in generating accurate code. It helped a lot. I saved time; the answer was straight to the point. The contextualization was the biggest gain for me"*.

Iterative refinement of generated codes through chat interaction. Given that StackSpot AI operates as a conversational agent equipped with an internal memory to record past interactions, users can engage with it in a manner akin to a chatbot. This feature of maintaining a history of previous conversations was initially viewed as interesting. It facilitates interactive code refinement and assists in honing code outputs to align precisely with specific project requirements, thereby enabling users to achieve more optimal coding solutions.

Support for repetitive and routine tasks through "quick commands". Quick commands are shortcuts offered by StackSpot AI, which developers could use to automate common software engineering tasks, such as creating tests, documenting code, or even asking the tool to explain a certain code snippet. Participants initially underestimated the impact of this feature, considering it a minor convenience for routine coding tasks. As they became more familiar with StackSpot AI, the collective sentiment shifted to view these quick commands as time-savers, greatly aiding in automating mundane aspects of coding and allowing them to focus on more complex tasks, assisting in improving code reliability and maintenance. One participant acknowledged the use of "quick commands" as a potential enhancement to their development workflow.

Time-saving by centralizing information access. Participants initially recognized StackSpot AI as a convenience by minimizing the need to alternate between various information sources. With continued use of StackSpot AI, they appreciated its efficiency in providing centralized access to essential information within the IDE, notably reducing development time and cognitive load. This was underscored by participant P1: *"I obtained the answer from the code StackSpot AI generated; I didn't need to go to the original knowledge source. In this case, it was accurate and helpful. [...] Even though the question I asked was a very simple example, StackSpot AI demonstrated that centralizing information in the IDE would be highly useful."*

5.3 Perception of productivity

After discussing the potential benefits of using StackSpot AI, we asked participants' opinions about its potential impact on their productivity.

⁷Note that we are reporting what our informants said, without discussing whether the concepts they used, e.g., productivity, are accurate or not.

Contextualized code snippets. A few participants affirmed that StackSpot AI indeed saved their time by providing quick and contextual code snippets during the study. The ability of StackSpot AI to quickly understand the context and deliver precise code snippets not only streamlined the coding process but also allowed users to focus more on creative and complex aspects of their work. When asked one participant mentioned: *“Yes, because within the context of each project, the knowledge base will become increasingly richer and will improve in generating responses.”*

Aggregating knowledge sources within the IDE. As mentioned among the benefits, participants specifically mentioned that StackSpot AI saved time by eliminating the need to search for knowledge sources because they are now “available” within the IDE. This feature was highlighted as a major time-saver, therefore influencing the perception of productivity of some study participants. For instance, one respondent mentioned: *“It StackSpot AI ‘saves time, as it’s a shortcut for accessing information. If done through conventional means, you’d have to search through a search engine, consult books, or find people with that information to assist you, which would certainly take longer.”* Another participant mentioned that by providing knowledge sources within the IDE, StackSpot AI could also reduce interruptions: *“I believe it can save our time, as it allows us to reduce interruptions when seeking specific information.”* In summary, by providing instant access to relevant knowledge sources directly within the tool, StackSpot AI enabled users to access necessary information or code samples without disrupting their workflow.

Productivity gains unlock only if users know how to use StackSpot AI. A participant commented that, like any AI tool, StackSpot AI only brings time-saving benefits if used correctly with refined prompts and proper settings. If used incorrectly, it could even lead to time wastage. For instance, one participant complemented the following: *“However, if used carelessly or by less experienced people, it may result in more work for the more experienced developers.”* This insight underscores the importance of understanding how to effectively interact with AI tools. Properly formulated queries and a clear understanding of StackSpot AI’s capabilities are essential to harness its full potential and avoid counterproductive outcomes.

Insufficient experience to evaluate. Finally, a few participants mentioned that the participation in the study was not enough to draw a definitive perception of the productivity gains of StackSpot AI, as one engineer highlighted: *“I don’t think I used it for enough time and in scenarios that would allow me to answer this question.”*

5.4 Challenges Encountered

This section highlights the various challenges encountered during the participants’ interaction with StackSpot AI. We divided these challenges into three groups: (i) challenges associated with the adoption of the Retrieval-Augmented Generation (RAG) technique, specifically the knowledge sources used; (ii) challenges associated with large language models in general; and, finally, (iii) other technical and user challenges associated with either UI aspects or user expectations. We present the challenges according to these groups.

Figuring out what is a good knowledge source. As mentioned in Section 3.1, knowledge sources are representative documents that enrich the prompts for RAG’s generation component, providing essential context for task development. Without these sources, responses from StackSpot AI would be less contextualized, resembling the answers from general-purpose coding AI assistants. Thus, identifying effective knowledge sources is vital for StackSpot AI’s performance. Our study, however, revealed that not all participants were able to understand what constitutes a (good) knowledge source. This was observed during group discussion about other kinds of knowledge sources they would use, based on their team context. While a few participants were able to give interesting examples (e.g., using a database schema as a knowledge source, and asking StackSpot AI to create SQL queries based on it), other participants were unable to come up with one single example. Furthermore, others gave examples that were not based on coding tasks, and a few participants even mentioned that they *“need to understand more about it, but were able to use it in a very basic way.”*

Knowledge Source Mixing Impact. During the focus group, it was mentioned that mixing several different knowledge sources affected the accuracy of the responses in certain cases, i.e., users found that the blending of information sometimes led to less accurate or relevant code suggestions, highlighting the need for better source management. This observation suggests that while having access to a wide range of sources can be beneficial, it also poses a challenge in ensuring that the information drawn from these sources is relevant and accurately integrated. One participant highlighted this issue as the following: *“Initially, I thought using various Knowledge Sources in the same workspace might not be a good practice. This led me to experiment with my own project. I combined a postal code API and various elements from the provided knowledge source. The result wasn’t great.”*

Inaccurate code suggestions and prompt refinement. Participants noted the necessity to refine and adjust prompts to obtain accurate responses. For instance, one participant mentioned that *“In some cases, they [the code suggestions] were not accurate; in others, they required many interactions and didn’t yield the expected result.”* Another participant added that *“It wasn’t as accurate; in my case when I entered it as ‘Go’ [the programming language] it generated generic things. It seems that it works better in Java.”*, revealing a potential bias towards more popular programming languages. This feedback underscores the importance of clear prompt formulation when interacting with AI tools. It also points to the potential need for iterative interaction, where initial responses serve as a starting point for further refinement to achieve the desired outcome. Furthermore, this result highlights how the participants required additional effort and understanding of how to effectively communicate with the AI, which was a learning experience for several participants.

Response Inconsistency: Our informants revealed that StackSpot AI sometimes provided varied responses to identical prompts. This inconsistency in output led to confusion among users and raised questions about the reliability of the tool in repetitive tasks. Although in the LLM literature, it is well-discussed that slightly different prompts could lead to different answers, users found this

experience awkward, potentially negatively impacting their trust in StackSpot AI.

Generating complex code structures. StackSpot AI faced difficulties in suggesting complex code like ready-to-use controllers. Participants expressed that while basic code generation was effective, the tool struggled with more sophisticated coding requirements. For example, one participant expressed this saying *“I tried to use the stack based on Spring, Java, Kotlin, and then play with the knowledge sources. However, despite my efforts, StackSpot AI was unable to generate the code with the endpoints. Instead of using controllers, it created methods in a main class that starts the SpringBoot app.”*

Inability to deal with custom languages. A participant proposes the ability to add custom language support for code snippets. The participant mentioned: *“When trying to include a snippet with a Cucumber Gherkin code, Gherkin does not appear in the list that defines what type of language the code refers to. Is it possible to register it?”* This may limit the ability to deal with specific/custom language and impact the outcomes for specific projects.

Conversation History Loss. A common frustration among participants was the loss of conversation history upon closing the IDE. This issue was particularly problematic for those working on complex tasks over extended periods, as it disrupted the continuity of their work and thought processes.

Missing User-Expected Features: Participants highlighted the absence of certain functionalities in StackSpot AI that they had anticipated. This gap in expectations versus reality suggested a misalignment between the tool’s capabilities and the users’ needs. Similarly, the absence of adequate plugin availability and support for other IDEs was mentioned as a drawback, which restricted the usability of StackSpot AI across different development environments, impacting its adaptability.

Initial Configuration and Onboarding Process. At the beginning of the study, participants struggled with the setup process of StackSpot AI, finding the integration of elements like workspaces, AI stacks, and knowledge sources challenging. This initial complexity was a significant barrier for many—particularly for those less experienced with such environments—indicating a need for a streamlined onboarding process. Indeed, a few participants were unable to correctly set up the environment, and thus did not actively participate in the full study. Still, one participant mentioned that *“[the code suggestions] could be better, but I believe it was because of how I configured the Knowledge Sources.”* This feedback shows that StackSpot AI’s effectiveness is significantly related to appropriately configuring the environment.

Technical Limitations. Participants frequently encountered technical issues such as timeouts and error messages (403 and 500 statuses). Additionally, some reported accessibility problems on specific machines (*“I couldn’t use StackSpot AI with the client’s machine. Does this mean that I was only able to run StackSpot AI on my personal computer?”* - P6). Another limitation mentioned was the impossibility of having a Git repository as a knowledge source hinders the ability to understand and work with new or existing projects, with all information available in their Git repositories.

In this context, a participant stated that: *“The possibility to add a complete Git repository as a knowledge source. It would help A LOT in adding a quick context of documentation or an application.”* These technical limitations hindered the smooth operation of StackSpot AI, affecting the overall user experience.

6 DISCUSSION

Reflecting on our results, we noticed that using a contextualized model to support developers will be beneficial for the company developers. This was evidenced by the benefits listed in Section 5.2.

On the Accurateness of Code Generation. Several participants highlighted that the code suggestions they received from StackSpot AI were both useful and accurate for their purposes, enabling them to simply copy and paste the generated code directly into their projects. For instance, a developer was able to quickly generate an integration with an internal API of a specific client by simply requesting it through StackSpot AI. Being able to use the provided code without additional modifications not only saved time but also demonstrated the tool’s capability to understand and address specific coding needs accurately. This exemplifies the potential time-saving benefits of the tool, showcasing its ability to automate and simplify complex tasks. This is in line with other studies [1, 4, 17], adding evidence related to the power of models fine-tuned for specific contexts. Participants also noted that some suggestions were not entirely precise, necessitating adjustments and refinement of the prompts. Refining prompts is a strategy used by other software developers using academic (see [18]) or proprietary tools (see [1]). This is a challenge associated with LLMs in general, but, as we discussed in section 5.4, we also identified challenges related to the usage of knowledge sources and other technical and user-related challenges.

Usability and Developer Efficiency. From a usability perspective, important results are highlighted. First, the ability to offer shortcuts (quick commands) has been shown to be beneficial to the developers, supporting what Barke et al. called the acceleration mode [3]. Our results indicate that the available tool did support that working mode. Second, StackSpot AI’s design, based on an interactive chat instead of focusing on code completion [3], also provided an opportunity for the iterative refinement of code suggestions as well as avoided interruptions [1, 4] similar to what has been observed by Ross et al. [16]. Finally, the possibility to focus solely on the IDE while seeking information has been shown to be important for software developers using AI coding assistants [1].

Having the right mix of knowledge sources is important. One important challenge that we observed is associated with knowledge sources. As mentioned in section 3.1, these sources are used in the RAG approach to enhance LLM-generated content by anchoring it in external knowledge sources. Therefore, it is not surprising that our informants reported different aspects associated with it including the difficulty in identifying good knowledge sources, the importance of properly configuring StackSpot AI with the knowledge sources, and finally the negative impact of mixing different sources. An interesting research avenue would be exploring the efficiency of different knowledge sources, and creating systematic

approaches to do so, supporting (semi-)automatic ways of optimizing the creation of contextualized coding assistants.

Challenges and Improvements in LLMs. We also highlighted several perceived challenges using StackSpot AI. It was interesting to find out that participants reported that the tool was able to generate accurate code snippets, that helped to neatly integrate internal APIs. Our study showed that it is possible to reduce the issues with incorrectness [13, 19] by creating a contextualized model to retrieve information from contextual knowledge sources. In contrast, the non-deterministic nature of LLMs [14] remains an issue, i.e., some of the challenges are inherent to LLMs, while other challenges are purely technical (conversation history loss, initial configuration, etc) and will be addressed in future releases.

Enhanced Generic Configuration Options for AI Coding Assistant. Since participants reported challenges with the initial configuration, one potential way to move ahead would be providing a more flexible and generic way of configuring StackSpot AI. The goal here is to make the setup more user-oriented, without the need to include a lot of information before the use of the tool. This comes with a trade-off, since when the settings are properly configured, the responses are very precise and save time—therefore, making it too flexible may hinder less experienced team members. The setup was idealized in a way to get the key configuration/customization items from the user, to make the code suggestions more reliable and precise. By making it less constrained, we may affect the accuracy of results. More investigation is required to understand how much flexibility is possible, without negatively impacting the outcome.

6.1 Limitations

This study, while extensive, has notable limitations. Firstly, our data collection involved a sizeable sample of practitioners using a contextualized coding assistant for the first time. However, this sample may not fully represent the broader population of software developers. Still, although the majority of the participants have previous experience with Generative AI tools, such as ChatGPT and GitHub Copilot, their experience with these tools might not naturally translate to the use of StackSpot AI, in particular, because they have to select and design representative knowledge sources, which is, by design, an important effort — which is not required by general-purpose tools.

Secondly, due to company policy recommendations against requesting gender information during the prestudy to avoid participant discomfort, our study did not gather this data. This omission restricts our ability to conduct comprehensive comparative analyses across different gender groups. Another limitation concerns the robustness of StackSpot AI, currently in its beta phase. Some challenges noted by participants might stem from insufficient testing rather than flaws in the underlying Large Language Model (LLM). This factor could adversely affect the overall user experience.

Given the nature of how the group discussion was conducted (as an online call with dozens of participants), we were unable to accurately identify the number of participants who mentioned a given benefit/limitation. As such, during our discussion section, we often refer to them as ‘many’, ‘a few’, ‘various’, and the like.

Moreover, our data analysis partially relied on AI-based tools. While these tools excel in summarization tasks [11], they do not adhere to stringent qualitative research methodologies. Consequently, while we successfully extracted representative quotes, the analysis’s rigor cannot be fully assured. To address this, two authors involved in the study reviewed and validated the AI-generated outputs. These authors suggested no additional items.

7 CONCLUSION

The recent flow of newly introduced AI coding assistants has unlocked developers’ potential, in a myriad of coding tasks. However, these coding assistants, when not trained taking into account the developers’ context (with their representative documents, coding styles, etc), might produce answers that although appear interesting at first, may not be as precise as developers’ need. Both academia and industry have recognized the need for more intuitive, conversational AI tools that could seamlessly integrate with existing IDEs, providing real-time, context-aware assistance.

In this work, we explored the use of StackSpot AI, a conversational AI tool, which is enriched with developers’ representative documents to generate more appropriate answers. We used the tool in a controlled setting with 62 practitioners. Our findings revealed that StackSpot AI could improve productivity and time efficiency. Participants appreciated its ability to quickly generate accurate code snippets and contextual code suggestions. However, the effectiveness of StackSpot AI was contingent on precise prompt formulation and optimal configuration of knowledge sources. Some challenges, like technical limitations and the need for better support in diverse IDE environments, were identified, highlighting areas for improvement. Other challenges are inherent to LLMs and require further AI research.

7.1 Future work

In *future work*, we plan to expand the scope of our research activities based on the insights gained from this paper. A key area of exploration will be to conduct longitudinal studies with developers who use StackSpot AI over extended periods. This will provide deeper insights into how prolonged use affects productivity, learning curves, and code quality. This will also allow us to understand how such a tool will impact developers’ work practices. Additionally, comparative studies involving other AI-assisted coding tools will offer a broader perspective on StackSpot AI’s unique strengths and areas for improvement. We also aim to investigate how StackSpot AI operates on a wider range of programming languages, assessing its adaptability and effectiveness across diverse coding scenarios. Further, exploring the impact of StackSpot AI on team dynamics and collaborative coding practices could provide valuable insights into its role in team-based development settings. Finally, delving into user customization and personalization aspects of StackSpot AI could reveal how tailored experiences influence developer satisfaction and tool efficiency. These research activities will collectively contribute to a more comprehensive understanding of AI-assisted coding tools in software development.

ACKNOWLEDGMENTS

We would like to thank the reviewers for their valuable comments. This work is partially supported by INES (www.ines.org.br), CNPq (grants 420406/2023-9, 442779/2023-2, 465614/2014-0, and 308623/2022-3), FAPESPA (053/2021), FACEPE (grants APQ-0399-1.03/17, PRONEX APQ/ 0388-1.03/14), CAPES (88887.136410/2017-00), and the National Science Foundation (grant numbers 2236198, 2303042).

REFERENCES

- [1] Anonymous Authors. 2024. “You’re on a bicycle with a little motor”: Benefits and Challenges of Using AI Code Assistants. In *International Conference on Collaborative and Human Aspects of Software Engineering (CHASE)*. 9.
- [2] Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, Quyet V. Do, Yan Xu, and Pascale Fung. 2023. A Multitask, Multilingual, Multimodal Evaluation of ChatGPT on Reasoning, Hallucination, and Interactivity. *arXiv:2302.04023 [cs.CL]*
- [3] Shradha Barke, Michael B James, and Nadia Polikarpova. 2023. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 85–111.
- [4] Christian Bird, Denae Ford, Thomas Zimmermann, Nicole Forsgren, Eirini Kalliamvakou, Travis Lowdermilk, and Idan Gazit. 2023. Taking Flight with Copilot: Early Insights and Opportunities of AI-Powered Pair-Programming Tools. *Queue* 20, 6 (jan 2023), 35–57. <https://doi.org/10.1145/3582083>
- [5] Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2023. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201* (2023).
- [6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgren Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).
- [7] Yan Chen, Sang Won Lee, Yin Xie, Yiwei Yang, Walter S Lasecki, and Steve Oney. 2017. Codeon: On-demand software development assistance. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 6220–6231.
- [8] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M Zhang. 2023. Large Language Models for Software Engineering: Survey and Open Problems. *arXiv preprint arXiv:2310.03533* (2023).
- [9] Patrick S. H. 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. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6–12, 2020, virtual*, Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). <https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html>
- [10] Welder Pinheiro Luz, Gustavo Pinto, and Rodrigo Bonifácio. 2019. Adopting DevOps in the real world: A theory, a model, and a case study. *J. Syst. Softw.* 157 (2019). <https://doi.org/10.1016/j.jss.2019.07.083>
- [11] Derek Miller. 2019. Leveraging BERT for extractive text summarization on lectures. *arXiv preprint arXiv:1906.04165* (2019).
- [12] Arghavan Moradi Dakkel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, and Zhen Ming (Jack) Jiang. 2023. GitHub Copilot AI pair programmer: Asset or Liability? *Journal of Systems and Software* 203 (2023), 111734. <https://doi.org/10.1016/j.jss.2023.111734>
- [13] Nhan Nguyen and Sarah Nadi. 2022. An Empirical Evaluation of GitHub Copilot’s Code Suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories (Pittsburgh, Pennsylvania) (MSR ’22)*. Association for Computing Machinery, New York, NY, USA, 1–5. <https://doi.org/10.1145/3524842.3528470>
- [14] Shuyin Ouyang, Jie M. Zhang, Mark Harman, and Meng Wang. 2023. LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation. *arXiv:2308.02828 [cs.SE]*
- [15] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2022. Asleep at the Keyboard? Assessing the Security of GitHub Copilot’s Code Contributions. In *2022 IEEE Symposium on Security and Privacy (SP)*. 754–768. <https://doi.org/10.1109/SP46214.2022.9833571>
- [16] Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael J. Muller, and Justin D. Weisz. 2023. 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, IUI 2023, Sydney, NSW, Australia, March 27–31, 2023*. ACM, 491–514. <https://doi.org/10.1145/3581641.3584037>
- [17] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems extended abstracts*. 1–7.
- [18] Frank F. Xu, Bogdan Vasilescu, and Graham Neubig. 2022. In-IDE Code Generation from Natural Language: Promise and Challenges. *ACM Trans. Softw. Eng. Methodol.* 31, 2, Article 29 (mar 2022), 47 pages. <https://doi.org/10.1145/3487569>
- [19] Burak Yetistiren, Isik Ozsoy, and Eray Tuzun. 2022. Assessing the Quality of GitHub Copilot’s Code Generation. In *Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering (Singapore, Singapore) (PROMISE 2022)*. Association for Computing Machinery, New York, NY, USA, 62–71. <https://doi.org/10.1145/3558489.3559072>