Unveiling the Potential of a Conversational Agent in Developer Support: Insights from Mozilla's PDF.js Project

João Correia
Pontifical Catholic University
Rio de Janeiro, RJ, Brazil
jcorreia@inf.puc-rio.br

Morgan C. Nicholson
University of São Paulo
São Paulo, SP, Brazil
nich.morgan@usp.br

Daniel Coutinho
Pontifical Catholic University
Rio de Janeiro, RJ, Brazil
dcoutinho@inf.puc-rio.br

Caio Barbosa
Pontifical Catholic University
Rio de Janeiro, RJ, Brazil
csilva@inf.puc-rio.br

Marco Castelluccio
Mozilla Corporation
London, UK
mcastelluccio@mozilla.com

Marco Gerosa
Northern Arizona University
Flagstaff, AZ, USA
marco.gerosa@nau.edu

Alessandro Garcia
Pontifical Catholic University
Rio de Janeiro, RJ, Brazil
afgarcia@inf.puc-rio.br

Igor Steinmacher
Northern Arizona University
Flagstaff, AZ, USA
igor.steinmacher@nau.edu

ABSTRACT

Large language models and other foundation models (FMs) boost productivity by automating code generation, supporting bug fixes, and generating documentation. We propose that FMs can further support Open Source Software (OSS) projects by assisting developers and guiding the community. Currently, core developers and maintainers answer queries about processes, architecture, and source code, but their time is limited, often leading to delays. To address this, we introduce DevMentorAI, a tool that enhances developer-project interactions by leveraging source code and technical documentation. DevMentorAI uses the Retrieval Augmented Generation (RAG) architecture to identify and retrieve relevant content for queries. We evaluated DevMentorAI with a case study on `PDF.js` project, using real questions from a development chat room and comparing the answers provided by DevMentorAI to those from humans. A Mozilla expert rated the answers, finding DevMentorAI's responses more satisfactory in 8/14 of cases, equally satisfactory in 3/14, and less satisfactory in 3/14. These results demonstrate the potential of using foundation models and the RAG approach to support developers and reduce the burden on core developers.

CCS CONCEPTS

• **Software and its engineering**;

KEYWORDS

Large Language Models, Conversational Agents, Developer Assistance, Software Development, Software Engineering

AIware '24, July 15–16, 2024, Porto de Galinhas, Brazil

ACM ISBN 979-8-4007-0685-1/24/07
https://doi.org/10.1145/3664646.3664758

ACM Reference Format:
João Correia, Morgan C. Nicholson, Daniel Coutinho, Caio Barbosa, Marco Castelluccio, Marco Gerosa, Alessandro Garcia, and Igor Steinmacher. 2024. Unveiling the Potential of a Conversational Agent in Developer Support: Insights from Mozilla's PDF.js Project. In *Proceedings of the 1st ACM International Conference on AI-Powered Software (AIware '24), July 15–16, 2024, Porto de Galinhas, Brazil.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3664646.3664758

1 INTRODUCTION

Open source Software (OSS) projects depend on open collaboration environments, in which a distributed community of developers and developers work in sync to develop the project [5, 6, 15]. For these projects to thrive, they must create a sustainable ecosystem in which communication between users and developers is constant and healthy [6, 9, 31]. As such, these communities often employ communication tools such as forums, chat clients, and issue trackers, enabling users and developers to more easily perform communication tasks, such as asking and answering questions that may arise throughout the development or usage of the project.

Furthermore, newcomers willing to join these projects have overcome several barriers to onboard [27], and rely on community resources to overcome them. This is often done through informal mentoring [12], a practice in which more senior developers volunteer their time to answer questions from newcomers. However, this practice does not scale, as the time and effort required from the project members are significant [12, 26], often to the detriment of other important project-related tasks.

Communities have used different ways to scale up the way they provide resources for users and newcomers, such as creating documentation that covers different facts [28]. However, it is known that the documentation in OSS projects is usually spread, disorganized, and lacking logical relationship [13]. As a result, communities continue to depend on humans to convey the information that is needed by newcomers effectively. Nevertheless, communities have continued to experiment with alternative ways to provide this information that are more efficient than utilizing human labor.

Foundation models, such as GPT-4 [2], have demonstrated their ability to provide accurate and satisfactory responses to a wide range of general inquiries [17, 21, 24, 33], and thus are being extensively utilized by developers to help them solve problems related to the projects they participate [19, 23]. Since these models can understand different types of content related to the project (e.g., source code, documentation, diagrams, etc.) they can efficiently organize information from disparate sources of information to provide information that helps new developers and users in answering their questions [21, 22].

However, most widely available tools utilized fixed, general, and sometimes outdated datasets as their primary source of information [3]. Since software projects have specific information, processes, and codebases —which are constantly evolving—it can be difficult to ensure that the information these tools provide is accurate [18]. Most of these tools are proprietary, providing limited experience in reporting how these foundation model-based assistants can help developers with project-specific questions, especially more specific questions requiring specialized knowledge.

To address this challenge, we introduce DevMentorAI, a conversational assistant that uses continuous integration to use project contextual information (reducing the risk of providing incorrect information). To accommodate for the specifics of different communities, DevMentorAI utilizes a modular approach, and thus can be easily integrated with different communication tools, such as forums, chat clients, collaborative development environments, etc.

To tackle the lack of experimental reports about how these conversational agents are helping developers, this paper presents a case study. In this study, DevMentorAI was applied to answer 14 real questions that were asked and answered by human developers in the chat room for `PDF.js`,[1] an OSS project managed by the Mozilla Foundation. A project expert from Mozilla (the main developer) was then tasked with evaluating how satisfied they were with each of the answers (*i.e.,* the human's and DevMentorAI's answers).

In summary, the expert perception was favorable towards DevMentorAI's answers in most questions (8 of 14 questions). In 3 questions, they were considered equivalent, and only in 3 questions, the expert was more satisfied with the human answers. To better understand these results, we also discuss individual cases and point out factors that may have influenced both the performance of DevMentorAI and the human, while also pointing at some factors that may have influenced the expert's evaluation.

Contributions: This paper has two main contributions: (i) we present DevMentorAI, a conversational agent that can help developers by utilizing real-time information, therefore helping reduce the time and effort expended by human developers in mentoring, and (ii) we report a case study in which DevMentorAI was utilized to answer 14 real questions asked by developers and users from a large open-source project (`PDF.js`). We then tasked a project maintainer from the project, with comparing how satisfied he was with the answers provided by the humans and by DevMentorAI.

2 RELATED WORK

This section presents related work on large language models used to support software development activities.

The development of Large Language Models (LLMs) has accelerated significantly over the past few years, marked by the introduction of models such as Gemini [30], BERT [10], and GPT-3 [7]. These models can generate human-like text, summarize content, and perform a wide range of natural language processing tasks. Recent work [17, 21, 24, 33] have been investigating the potential of LLMs in supporting or acting as assistants for SE activities. Dakhel *et al.* [17] explore Copilot's effectiveness in providing correct, efficient, and reproducible solutions for fundamental programming tasks, and assess whether Copilot's solutions are competitive with those created by humans. Sallou *et al.* [24] emphasizes concerns such as closed-source models, data leakage, and the reproducibility of research findings involving LLMs. Moreover, Balfroid *et. al.* [4] explored the potential of using LLMs for generating annotated code tours to help the onboarding process of new developers in software projects. Their findings show that the LLM, while able to generate code and present solutions, often gives vague information or gives the same content present in the documentation.

Pudari *et al.* [22] studied the current capabilities and limitations of AI-supported code completion tools like GitHub's Copilot. It explores how these tools deal with programming idioms and code smells. The study finds that Copilot often does not suggest the most idiomatic or best practice code, indicating areas for future improvement in AI-supported programming tools. Moreover, Sauvola *et al.* [25] investigated the impact of generative AI on software development, analyzing the opportunities, challenges, and ethical concerns surrounding AI in enhancing productivity, and automation in software development. The study emphasizes the need for new tools and guidelines to navigate the world of generative AI to better understand its role in software development practices and operations. Finally, Pinto [21] explores the use of ChatGPT for grading and providing feedback on open-ended questions within technical training contexts. Their findings suggest that ChatGPT can identify semantic-related details in responses and it is generally aligned with expert corrections, indicating its potential use in the feedback process for open questions in educational environments.

Our study differs from prior research in two main ways: (i) we designed a conversational assistant that supports interfacing with different generative models (currently GPT-3.5 Turbo). Using these models, we employed a Retrieval-Augmented Generation (RAG) approach, focusing specifically on documentation and source code from the target project. As such, our assistant is specialized in answering questions related to the project (beyond code); (ii) we assessed the quality of the responses of our custom assistant by comparing them with actual real-user answers, which were subsequently evaluated by a project expert.

3 DEVMENTORAI

In this section, we present the details about DevMentorAI. The source code is available on our replication package [1].

3.1 Requirements

We designed DevMentorAI as a conversational assistant aimed at aiding developers by offering relevant responses to project-specific questions. We focused on creating a tool that would enable compatibility with various platforms, like chat clients, forums, and issue

[1] https://github.com/mozilla/pdf.js

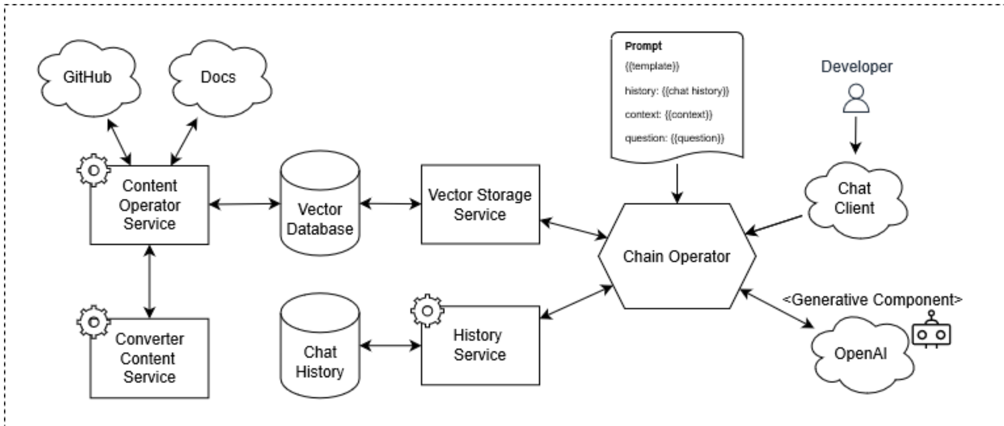

Figure 1: DevMentorAI's Pipeline.

trackers, while maintaining a swift response time to seamlessly integrate into the developer's workflow. Additionally, a core aspect of DevMentorAI's functionality is providing accurate and current data, ensuring that developers have access to the latest information relevant to their projects.

3.2 Architecture

Given the goals presented above, we opted to design DevMentorAI by leveraging the hexagonal architectural framework [29]. By using this architecture, DevMentorAI's execution follows the pipeline summarized in Figure 1.

The use of the hexagonal framework simplifies the development of new adapters for different data sources (*e.g.,* scrapers, PDF readers, git)—via the content operator and converter content services (left of Figure 1)—and enables smooth integration with different communication clients (*e.g.,* GitHub issue tracker, Matrix Chat, Discord)—which can be plugged to the chain operator (right of Figure 1). This design enables extensibility in terms of inputs and outputs, and easier maintenance for different projects and contexts. This novel way of building a contextual agent makes DevMentorAI's architecture a good fit for several projects, regardless of the source of documentation and communication channels they use.

Next, we will discuss the individual components developed for DevMentorAI, shown in the figure.

3.2.1 Vector Storage Service. The first step to creating a contextual agent is collecting the information from the data sources. For retrieving the up-to-date information that DevMentorAI utilizes to answer questions, we opted for the Retrieval-Augmented Generation (RAG) strategy [16]. As such, we developed a component that ingests the projects' documentation and source code for the assistant's usage. This occurs in two steps. First, this component processes the artifacts from the project, and second, it then identifies similarities between the questions and the project context (via a vector space). We will detail each of these steps as follows.

Obtaining the raw sources. DevMentorAI utilizes the *Content Operator Service* to capture the target project source code and documentation (by integrating with git/GitHub) and other documentation web pages (through web scraping). When needed, a universal text format converter is utilized (*Content Converter Service*, which currently utilizes *Pandoc*) to standardize raw text into Markdown. The processed content is then broken down into smaller chunks for targeted analysis of source code or documentation relevant to specific questions. This approach avoids comparing entire documents or code modules. Programming language and text semantics are respected during this stage to maintain coherence and syntax.

Content Vectorization. Each excerpt of raw content is then transformed into a vector and stored in a database, thereby updating the project's context. This vectorization process is done to optimize searching for documents or source code chunks similar to the question being asked (*i.e.,* providing context for an answer).

3.2.2 Generative component (OpenAI). DevMentorAI's architecture was planned to accommodate different generative AI tools—we call them "Generative components" in the scope of our architecture. For the sake of this study, we utilized OpenAI's *GPT-3.5 Turbo* with a 4K token context window as our core generative model. Our selection of OpenAI's GPT model was driven by its status as the leading language model available during our study. Additionally, we opted for *GPT-3.5 Turbo* over *GPT-4* due to monetary constraints. The communication between our assistant and the generative model was executed using the reference API provided by OpenAI[2].

3.2.3 History Service. To increase the richness of the contextual information input provided to the generative model, we provide a rolling history of the conversation to the prompt. To do so, we adopted the `ConversationBufferMemory` [3] construct from LangChain [8]. This construct was chosen given its ability to provide a history without the necessity to first use Generative AI to summarize historical data.This strategy also reduces the response time and increases the accuracy of the answers.

3.2.4 Chain Operator. On the other side of the architecture, the user posts questions that are handled by a conversation chain, which is operated by a "chain operator." When the user asks a question, this operator is invoked and passes the question to the pipeline. The operator has a pre-set prompt template that needs to be filled with certain variables, as in the top-left of the example in Figure 1. This prompt has details about the persona, which is information

[2]https://platform.openai.com/docs/api-reference
[3]https://python.langchain.com/docs/modules/memory/types/buffer

related to how the assistant should behave. The prompt also contains relevant details the assistant should consider to generate the answer. This template requires three pieces of information: *history*, which is the history of previous interactions, *context*, which is the knowledge relevant for answering the question (*i.e.,* documents and source code chunks similar to the question asked), and, of course, the *question* posted by the user.

The *history* is filled by the *history service*. It keeps a rolling window of previous interactions with the user. The *context* is filled by asking the *vector storage service* to search for chunks that are similar to the vector representation of the question asked. Once the relevant vectors are retrieved, the full contents of these chunks are restored from the metadata to fill the context variable.

4 METHODOLOGY

The goal of this study is to assess how DevMentorAI answers compare to those written by humans in a project-specific context. So, we conducted a case study using real past questions in a development chat channel and analyzed how DevMentorAI performs in those questions. Thus, our research question is:

> **RQ:** How does DevMentorAI compare to humans when responding to project-specific questions?

4.1 Study Design

We conducted four main methodological steps to answer our research questions, described in the following sections.

4.1.1 Project Selection. The Mozilla Corporation maintains multiple open-source projects in several domains, most of which could be suitable for our investigations. To assess our approach we selected the project to conduct our evaluation based on three criteria: (i) the discussions on the project should be openly available, (ii) our collaborators at Mozilla should have contact with the engineers of the project, and finally (iii) the project should be ingested by our model in a reasonable time.

Here is some rationale supporting our criteria. First, we needed access to the discussion channel of the project. Therefore, we limited the projects' universe to those with open discussions on Matrix—the Mozilla platform for discussions. Second, since our collaborators were supposed to evaluate the DevMentorAI answers for real questions, we restricted the projects to those in which we could reach the engineers of the project. Third, ingesting the project's repository artifacts takes a long time, depending on the project size, so we opted for small or medium size projects. Therefore, observing these criteria, we selected the `PDF.js` as the subject for our study.

4.1.2 Ingestion of project's artifacts. Our assistant ingested technical documentation and source code extracted from `PDF.js` to conduct this case study. We collected all the artifacts from the project mozilla/pdf.js hosted on GitHub. It is worth mentioning that our assistant supports ingesting source code from several languages in addition to markup files. Aiming to standardize the file's vector computing, we employed Pandoc [20] to convert the markup files such as *.html* to *.md*. In general, for the `PDF.js`, we ingested source code files, wiki pages, and web pages comprising files in the following formats: *js*, *jsx*, *mjs*, *cjs*, *md*, *html*, *pdf*, etc. To assess the effectiveness of artifact ingestion, we conducted some preliminary isolated tests, which produced contextualized outcomes when asking questions about the `PDF.js`.

4.1.3 Selecting the Questions. With the subject project chosen, we started gathering the conversations in the chat channel, which project developers and users use. We accessed the `PDF.js` Matrix room[4] and exported the chat history leveraging a built-in functionality. We collected all messages and their metadata until January 16, 2024, in a JSON format.

Since our interest was to obtain textual questions and their answers, we made an initial data processing using Python scripts. We created as output a spreadsheet containing (i) the message timestamp, (ii) whether the message was a reply from a previous message, and (iii) its textual content. This process resulted in a spreadsheet containing 3492 messages. Next, one author manually inspected each message to determine whether that was a question with a proper human response. This process was conducted by one researcher and validated by a second.

During the manual analysis, we identified several questions throughout the chat history. However, the preponderance of these questions proved unsuitable for inclusion in our evaluation. Initially, we discarded cases where questions were posed, and other users requested more information, leading to threaded discussions to completely understand the question's context. Additionally, we disregarded unanswered or questions that required access to external resources. Finally, non-technical queries were excluded from consideration. Below, we have an instance of an ignored question and the reasons presented above.

> **Question discarded in the manual analysis**
>
> Hi all, I'm trying to implement the pdfjs on a certain project. I was able to implement the example on the official site that uses the pdfjs-dist script, but I am a little confused as to how I can implement this example: https://mozilla.github.io/pdf.js/web/viewer.html in a similar manner?

We ignored the aforementioned example predominantly due to its lack of answers. Furthermore, accessing the link to check the code example provided in the question is necessary to gain context and provide an accurate answer. Since the model did not have access to external artifacts, that was another factor to classify that question as unsuitable. After the manual analysis, we ended up considering 14 questions and their respective answers for this preliminary analysis. It is worth noting that some users asked questions using multiple messages. In these situations, we concatenated the messages to conceive the question. The same approach was adopted for the answers. Still, follow-up questions were not considered.

4.1.4 Expert Assessment. After selecting the fourteen questions, we submitted each of them to DevMentorAI. We implemented an integration with Discord [11] and performed the following steps for each question. First, we started a new chat, which created a new and empty context for the conversation. Next, we copied the question we identified in the chat and submitted it for the assistant consideration. DevMentorAI processed the data going through

[4] https://chat.mozilla.org/#/room/#pdfjs:mozilla.org

our vectorial database and sending the contextual prompt to the foundational model. Finally, we received the answers.

We structured the data from the previous steps in a spreadsheet with three columns. The first column contained the question posted; the second had the human answer as posted in Matrix; and the third presented the responses generated by DevMentorAI. Subsequently, we added two more columns to that spreadsheet to enable the specialist assessment, which we describe in the following paragraph.

Since we conducted this study in collaboration with the Mozilla Corporation, the answers were assessed by an expert with vast experience in programming (15 years), 4 years experience on the target project (*i.e.,* PDF.js), and that is currently the project's main developer. We asked this expert to answer the following questions for each question: "How satisfied are you with the human answer?" and "How satisfied are you with the bot's answer?". Both questions enabled answers following a 5-point Likert scale, as follows: *not at all satisfied* (0), *slightly satisfied* (1), *moderately satisfied* (2), *very satisfied* (3), and *completely satisfied* (4).

To further understand the context and details of the conversations, we explored the sequence of the chat analyzing the messages that followed the question and answer on Matrix. To do so, we went back to Matrix and manually analyzed the messages exchanged about the topic, providing us with further evidence of the satisfaction and usefulness of the answers provided in the chat.

5 RESULTS AND DISCUSSIONS

In this section, we present the results of our study to characterize the DevMentorAI performance in answering project-specific questions and the factors that influence their responses.

5.1 Overall Performance

As described in Section 4.1.4, we compared the performance of DevMentorAI with the actual humans' answers when addressing questions regarding `PDF.js`. For that, a `PDF.js` expert from Mozilla gauged its satisfaction with the answers the humans and our assistant provided. This expert has extensive experience in programming and on `PDF.js`, featuring as the project's main developer.

Figure 2 summarizes the results from the comparative analysis. The x-axis shows the questions, while the y-axis provides the ratings for answers. The performance evaluation yielded the following results: (i) DevMentorAI surpassed the human in 8 questions; (ii) in 3 questions, both exhibited comparable performance; and (iii) in 3 questions, the human outperformed DevMentorAI. One can observe that in all the cases in which the expert was more satisfied with the human's answer, the difference was at least two points on the Likert scale. On the other hand, DevMentorAI had this degree of superiority in 5 cases (out of the 8 cases mentioned in (i)).

5.2 Discussion About Individual Cases

In this section, we discuss the perceived performance of DevMentorAI in specific cases. We also point at factors that may have influenced both how DevMentorAI responded and how the expert evaluated the case. These examples will be divided into three categories. First, the cases in which the response from DevMentorAI was perceived as better than that from the human. Second, the ones in which the response from the human was perceived as better. Third, those cases in which they were perceived as similar. Although we explore exemplary cases for each category, all the questions and answers are available in our replication package [1].

5.2.1 ***DevMentorAI outperforms humans.***

This scenario was observed in 8 out of the 14 questions analyzed. In this section, we present three exemplary cases in which the expert ranked DevMentorAI's answer as more satisfactory than the human's answer.

Question 11

Hi, I have a very large pdf in size (200mb) and I want to download it faster. How to do that in PDF.js viewer?

1. Pdf is linearized.
2. Using rangeChunkSize (1024 1024).

But it is downloading chunk in sequentially but I want them to be downloaded parallelly. Is there any configuration available to control the number of parallel requests?

Human's Answer:

Download it and use an other viewer.

DevMentorAI's Answer:

No, there is no configuration available in the PDF.js viewer to control the number of parallel requests for downloading chunks of a large PDF file. The viewer automatically handles the downloading and rendering of the PDF file, and the number of parallel requests is determined by the browser's network stack.

In Question 11, someone asked if they could use the project in a configuration that is not supported. The human who answered provided a very short and direct answer, which one could perceive as ungentle or even uncaring, due to the dismissive way it was phrased. By analyzing DevMentorAI's answer, we observe that it explains to the participant the specifics of why it was not possible. Consequently, the expert evaluated that they were *not at all satisfied* (0) with the human's answer, and *very satisfied* (3) with DevMentorAI's. When we went in-depth into the conversation, we observed that the person asking the question provided more information after getting the rough answer, and the same respondent posted other rough comments, including an image of Taylor Swift with the following excerpt of her song: "I will never shut up about."

Question 12

Is there a pdf.js associated inspector-like tool for viewing pdf internals? I recall seeing something like this before.

Human's Answer:

yep: http://brendandahl.github.io/pdf.js.utils/browser/.

DevMentorAI's Answer:

Yes, there is a PDF Object Browser tool available for inspecting the internal object structure and viewing raw values of a PDF document. You can access the PDF Object Browser using this link: https://brendandahl.github.io/pdf.js.utils/browser.

In Question 12, the user asked for a tool included in the project. The human responded precisely, with a short confirmation and a link to the tool the user was looking for. DevMentorAI also replied

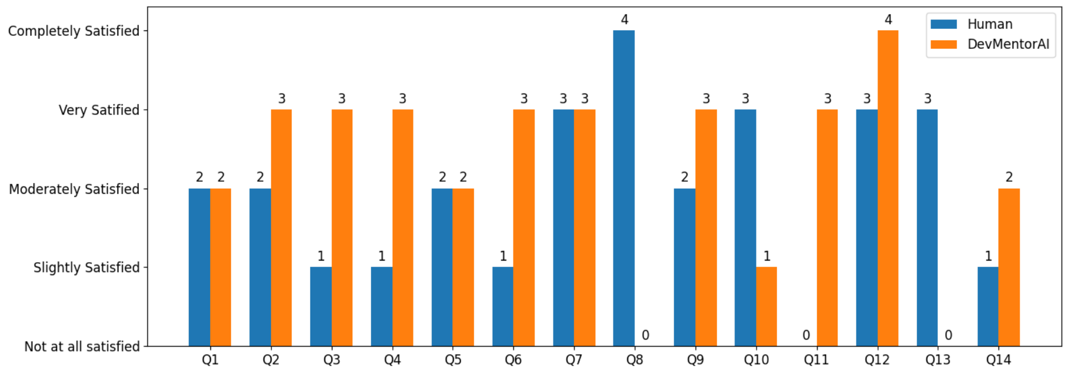

Figure 2: Expert ratings for questions in a Likert scale.

with the same link but briefly described the tool. The expert was satisfied with both answers and slightly preferred DevMentorAI's answer. They reported being *very satisfied* (3) with the human response and *completely satisfied* (4) with DevMentorAI's response.

We can observe that, in both cases presented here, the humans provided short answers, with poor context or explanations. This behavior may have multiple possible causes, including stress and lack of time. Moreover, this problem may have varying levels of severity. For instance, responses were perceived as ungentle or even toxic in the first question, but acceptable ones as the second one. This kind of behavior shows a problem inherent to human responses: the human might not always be fully available to provide a complete response. In these cases, we observe an advantage that is inherent to assistants such as DevMentorAI. Not only they are always available to answer the user but also provide an answer to the full extent of content they have access.

Question 3

Hello! I was trying to switch to reader view on FF on a badly typo-graphed PDF. Can this be done somehow? if its not a feature already, where can I suggest it?

Human's Answer:

I don't know how *technically* feasible it is, but I just got to https://www.supremecourt.gov/opinions/20pdf/18-956_d18f.pdf actually that's not a bad idea :) beyond the fact that it's all white space and small font, I get a lot of light sensitivity so prefer a dark background (which reader mode allows) You can suggest it by filing a bug on bugzilla.mozilla.org

DevMentorAI's Answer:

To switch to reader view in Firefox for a PDF document, you can follow these steps:

1. Open the PDF document in Firefox.
2. Click on the ""Reader View"" icon in the address bar (it looks like an open book).
3. The PDF will be displayed in a simplified, distraction-free reader view.

If you have any suggestions or feature requests for Firefox, you can submit them to Mozilla's Bugzilla platform. Here is the link to file a new bug report: https://bugzilla.mozilla.org/enter_bug.cgi?product=Firefox

Make sure to provide a clear description of your suggestion and any relevant details that can help the developers understand and evaluate your idea.

Question 3 illustrates an inherent advantage that virtual assistants can have over humans. In this case, the human was not aware of a feature that was already implemented in the software. Therefore, they only instructed the user on how they could suggest this feature to the development team. On the other hand, since the virtual assistant has access to several sources of up-to-date information about the project, it was aware of the existence of this feature, thus it both instructed the user on how to use the feature and how to suggest new features (the user also requested this information). As such, the expert was slightly satisfied (1) with the human's answer, and very satisfied (3) with DevMentorAI's answer.

5.2.2 ***DevMentorAI performs as good as a human.*** Here, we explore two of the three cases in which the expert was equally satisfied with the answers from the human and DevMentorAI.

Question 7

Hi folks, If I have a PDF that's 100MB and 100 pages long, and I want to direct a user directly to page 50 for example. Can PDF.js load that page only without fully loading the entire PDF? Or load only that range of pages...

Human's Answer:

If your sever supports range requests, then yes pdf.js should be able to only load part of the pdf.

DevMentorAI's Answer:

PDF.js can load specific pages or a range of pages from a PDF without loading the entire document. This is achieved through the use of HTTP Range Requests, which allow for fetching only the required portions of the PDF needed for rendering the visible pages. However, the ability to utilize HTTP Range Requests depends on browser support and the web server returning the appropriate headers. If these conditions are met, PDF.js will automatically start fetching the necessary portions of the PDF for rendering, allowing for a more efficient and faster loading experience.

In Question 7, both the humans and DevMentorAI provided answers with very similar content. However, one can quickly notice that the answer from the agent is more verbose, at least 4 times bigger than the human's. This is similar to what we observed in the previous section, with DevMentorAI providing more details and explanations about the answer. The expert reported to be *very satisfied* (3) with both answers. We believe that DevMentorAI's answer can benefit less experienced people or those with a comprehensive information processing style.

Question 5

Hey everyone! I'm having a frustrating issue where I'm trying to allow PDFs to be previewed in browser. I'm including them via URL in an 'embed' or 'object' HTML tag currently and while it works after installing the browser extension, it still downloads every PDF file that gets previewed on the page. Is there a way to prevent that?

Human's Answer:

To preview anything, it first needs to be downloaded. Nobody reads online. All content is downloaded and read locally from browser cache.

DevMentorAI's Answer:

No, there is no built-in way to prevent PDF files from being downloaded when previewed in the browser using an 'embed' or 'object' HTML tag. These tags are used specifically for embedding content in web pages and allow users to download the file if they choose. If you want to restrict the downloading of PDF files, you would need to implement custom logic on the server-side or use other methods like DRM (Digital Rights Management) to control access to the files.

For Question 5, both the human and DevMentorAI acknowledge that it is not possible to perform the action requested by the user. Once again we see that the human is more objective and direct than the agent. The expert reported to be *moderately satisfied* (2) with the answer. This may mean that the expert believes that the answers are not complete in any of the cases. For the human it may represent a shortage of knowledge, and for DevMentorAI, a lack of appropriate and accurate information in the project repository. By investigating the chat logs, we see that the conversation kept moving, and, at one point the human mentioned "*Sorry I am not expert*," while they kept trying to sort out the program together.

5.2.3 *DevMentorAI was outperformed.*

Finally, we discuss two of the three cases in which the answer from the human was more satisfactory than the ones from DevMentorAI. One important observation before we dive into the details, is that in these cases the answer from the agent received really low scores. The expert was not satisfied at all (zero score) in two cases.

Question 8

Hi, I wonder how to use the zoom commands, if I want to zoom into a pdf say 400 and define a left and top offset? https://mozilla.github.io/pdf.js/web/viewer.html?file=compressed.tracemonkey-pldi-09.pdf&#zoom=300,10,1 brings a zoomed version with a left offset but jumps to the second page. What do I have to change to get a top offset of 200px?

Human's Answer:

In the pdf coordinate system 0,0 is the bottom, left of a page.

DevMentorAI's Answer:

To modify the zoom commands in the PDF.js viewer to achieve a top offset of 200px, you can use the 'zoom' option with the desired values. The format for the 'zoom' option is '[zoom],[left offset],[top offset]'. Here is an example URL that demonstrates how to set the zoom level and top offset: https://mozilla.github.io/pdf.js/web/viewer.html#zoom=200,0,200. In this example, the zoom level is set to 200 and the top offset is set to 200px. You can adjust the values as per your requirements. Please note that this modification will only affect the viewer for the current session and will not permanently change the default behavior of the PDF.js viewer.

Question 8 is a complex case. As in the previous cases, the answer from the human is very short and simple. It just points out to the participant that the $0, 0$ position is the bottom left. The expert is *completely satisfied* (4) with this answer, understanding that the user who posted the question would be able to figure out how to go to the specific place in the PDF. Interestingly, DevMentorAI gives misleading information, saying that *"The format for the 'zoom' option is '[zoom],[left offset],**[top offset]**'."* Notice that the two last parameters here point to the **top**-left as the 0,0 position, while the human pointed to the **bottom**-left. This caused the dissatisfaction (zero score) of the expert. However, when analyzing the document presented in DevMentorAI's answer and the thread on Matrix, we see that (i) DevMentorAI found the appropriate resource and presented the information as provided in the documentation; and (ii) the information was incorrect on the documentation, as per the follow-up from the human respondent: *"hmm...I think that's wrong [...], but it should be changed or described in the wiki as well...* Due to this divergence—caused by the issue with the misleading information in the documentation—the human reported to be *completly satisfied* (4) with the human answer and *not at all satisfied* (0) with DevMentorAI's answer.

Question 10

Hi everyone, I'm running into a bit of an issue with PDF.js and chrome. I'm not sure I've got enough information for a true github issue, but I wanted to drop it here to see if you might have any insight into the issue.
In the image above you can see a sample of the problem: On my coworkers machine PDF's are often rendered like this, with random bits and pieces of the text characters missing. This seem to happen on his machine and at least one other but not on mine, and it only happens in Chrome and in Edge but not when using Firefox.
It seems to happen to many, but not all, the PDF's, and it isn't consistent: a page reload can cause the anomalies to end up entirely somewhere else in the document. I have attached an example PDF that exhibits the problem for my coworker.
This goes wrong in our own application and in the slack PDF viewer (which seems to use PDF.js), but in the PDF.js demo page (opening the example PDF) does work correctly. https://mozilla.github.io/pdf.js/web/viewer.html
I initially blamed my coworkers machine, but as we have now also seen it on another machine it is suspicious. Have you every seen anything like this? I'm pretty stumped. Any insight on this matter is greatly appreciated!

Human's Answer:

Maybe you should try to test in enabling/disabling hardware acceleration in Chrome I tried myself on Chrome on Windows 11 with enabled/disabled 'Accelerated 2D canvas' and it works well.

DevMentorAI's Answer:

I'm sorry, but I don't have access to personal experiences or encounters. However, it is possible to encounter rendering issues with PDF.js in different browsers. If you are experiencing this specific issue, it would be best to report it to the PDF.js development team with the necessary details such as the PDF file, browser version, operating system, and any error messages you may have encountered.

In Question 10, a developer contextualized the issue explaining that they are experiencing rendering problems with `PDF.js` and Google Chrome on particular machines and asked for insights into what could be causing the problem. The human answered with something that seemed like a personal experience. They guided the user to turn off the hardware acceleration in Google Chrome to solve the problem. In contrast, DevMentorAI answered that it cannot access personal experiences (given the personal question asked: "*Have you every [sic!] seen anything like this?*"). The expert classified the human's answer as *very satisfied* (4) and the DevMentorAI's answer as *slightly satisfied* (2). The satisfaction with the human answer goes beyond the expert, the user replied to the human in the thread saying" "*I tried myself on Chrome on Windows 11 with enabled/disabled 'Accelerated 2D canvas' and it works well.*"

6 LIMITATIONS AND DESIGN TRADE-OFFS

In this paper, we presented a case study to assess how satisfactory the answers of a conversational agent would be in the context of an OSS project. Our decisions on the study conception may have an impact on the results. First, we had to choose a subject project to run the study, which is a key decision in a case study. While many factors may influence this decision, we focused on finding a project maintained by Mozilla (a partner on this research) with experts on their reach, and that was not too big to ingest into our model for our preliminary evaluations. We acknowledge that our research results are influenced by the specific context of our case study (`PDF.js`) and that it limits the generalizability of our results. However, we preferred to keep more control and work on a single and smaller project, supported by the maintainer, which entitled us to be more careful in the data collection and better understand the context.

A second point regards evaluation from the expert. We could have conducted in-depth interviews or collected multiple opinions to get stronger evidence. Once again, considering the factors that led us to decide, we wanted to guarantee that we would get appropriate responses that would not require too much time or effort from the expert. Still, it is well known in the literature that recruiting is not an easy task in Software Engineering, especially when we deal with OSS [14, 32]. Therefore, since we are collaborating with Mozilla, and we had access to an expert, we preferred someone to provide us with accurate feedback.Although we would have a more in-depth understanding using other methods, we are very comfortable that the input received is reliable and accurate, given the expertise and willingness to help from this person.

We also had to decide on a specific LLM to use. While this may pose a threat, we used OpenAI's *GPT-3.5 Turbo*, since GPT was the state-of-the-art LLM at the moment of the study, and we opted for *3.5 Turbo* instead of *GPT-4* due to budget constraints. Even using a previous version, the results proved to be very encouraging. Still related to the LLMs, we acknowledge that they are subject to continuous updates and modifications, and threaten the consistency and replicability of our results. Nonetheless, in contrast to related studies, large language models (LLMs) rapidly evolve and enhance their models daily. Consequently, we anticipate that the quality of responses may improve progressively over time.

Some decisions made during the data curation process are also worth discussing. Given the context of the project under study and the choice of the LLM used, we had to dismiss questions that embedded images. This is an important decision because `PDF.js` is a PDF rendering tool, and screenshots and PDFs were part of a non-negligible number of messages. Although we missed some questions, our sample size was not too small and enabled us to understand different aspects of DevMentorAI performance. In future work, we plan to use Foundation Models that read images and documents to understand how they improve the answers.

We acknowledge that DevMentorAI answers may be limited for some lack of context. DevMentorAI had no access to the complete chat history. Therefore, any contextual information provided before those messages used to compose the question was not accessible by the agent. Moreover, since the Matrix channel is open, the human's answers might not be from developers from `PDF.js`, but from users. This may have resulted in suboptimal answers coming from them as well. However, this does not pose a threat since the answers analyzed are the ones that were actually posted to the questions.

Another point is that we did not control documentation in terms of accuracy and up-to-dateness of the information ingested. There may be cases in which the answers could be improved. Although these points might have influenced the quality of the output, our results were very satisfactory.

Still, two decisions related to the design and implementation of DevMentorAI may pose limitations. First, we decided to ingest data available on GitHub to our conversational agent. There may be more documentation outside of GitHub boundaries that were not considered. This may have reduced the accuracy of our responses. However, once again, the preliminary assessment is very encouraging, and this may be included in the following steps of this research. Second, we used cosine distance to compare the query and the vectors stored in our database as a step to create the prompt sent to the LLM. Although we acknowledge that other options would offer different results, we opted to keep a conservative approach to assess how the agent would perform with these conditions.

7 CONCLUSION

This paper introduces DevMentorAI, a conversational assistant that uses the Retrieval Augmented Generation (RAG) architecture to facilitate developers' interactions with software projects by integrating project-specific data. By doing a case study of the `PDF.js` project, DevMentorAI proved its capability to provide answers that, in most cases, were better than the ones from human participants. Our goal was to, by integrating project-specific data, assess if DevMentorAI could make the onboarding process smoother for new developers and automate responses to project-specific questions.

Moreover, this study highlights the importance of integrating detailed and context-specific information into conversational agents to improve their effectiveness. As foundation models continue to evolve, their integration into various aspects of software engineering and development promises to enhance productivity and facilitating processes. The findings from this study not only demonstrate the feasibility and utility of such models in practical applications but also pave the way for future research.

For future work, we aim to explore its application across a wider range of software projects to understand its adaptability and impact on developer productivity in diverse development environments.

ACKNOWLEDGEMENTS

This work was supported by the National Science Foundation grants 2236198, 2247929, and 2303042. Also, CAPES: 88887.899310/2023-00, 88887.900069/2023-00 and 88887.915794/2023-00. CNPq: 140770/2021-6, 141180/2021-8, 140771/2021-2, 315711/2020-5, 141276/2020-7 and 141054/2019-0. CAPES/PROCAD: 175956. CAPES/PROEX: 88887.373933/2019-00. FAPERJ: 200.510/2023, 211.033/2019 and 202.621/2019. Finally, we thank the Mozilla PDF.js maintainer for analyzing the answers and giving their feedback.

REFERENCES

[1] 2024. Replication Package. https://github.com/RESHAPELab/DevMentorAI-PDF.js-replication/. https://doi.org/10.5281/zenodo.11211921 Accessed: May 17, 2024.

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[3] Amos Azaria, Rina Azoulay, and Shulamit Reches. 2024. ChatGPT is a Remarkable Tool—For Experts. *Data Intelligence* 6, 1 (02 2024), 240–296. https://doi.org/10.1162/dint_a_00235

[4] Martin Balfroid, Benoît Vanderose, and Xavier Devroey. 2024. Towards LLM-Generated Code Tours for Onboarding. In *Workshop on NL-based Software Engineering (NLBSE'24)*.

[5] Magnus Bergquist. 2020. Open source software development as gift culture: Work and identity formation in an Internet community. In *New Technologies at Work*. Routledge, 223–241.

[6] Scott Brisson, Ehsan Noei, and Kelly Lyons. 2020. We Are Family: Analyzing Communication in GitHub Software Repositories and Their Forks. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 59–69. https://doi.org/10.1109/SANER48275.2020.9054834

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[8] Harrison Chase. 2022. LangChain. https://github.com/langchain-ai/langchain. Accessed: April 3, 2024.

[9] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work* (Seattle, Washington, USA) *(CSCW '12)*. Association for Computing Machinery, New York, NY, USA, 1277–1286. https://doi.org/10.1145/2145204.2145396

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805* (2018).

[11] Discord. [n. d.]. Discord. https://discord.com/. (Accessed on 05/14/2024).

[12] Zixuan Feng, Amreeta Chatterjee, Anita Sarma, and Iftekhar Ahmed. 2022. A case study of implicit mentoring, its prevalence, and impact in Apache. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Singapore) *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 797–809. https://doi.org/10.1145/3540250.3549167

[13] Felipe Fronchetti, David C. Shepherd, Igor Wiese, Christoph Treude, Marco Aurélio Gerosa, and Igor Steinmacher. 2023. Do CONTRIBUTING Files Provide Information about OSS Newcomers' Onboarding Barriers?. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (San Francisco, CA, USA) *(ESEC/FSE 2023)*. Association for Computing Machinery, New York, NY, USA, 16–28. https://doi.org/10.1145/3611643.3616288

[14] E. Kokinda, M. Moster, J. Dominic, and P. Rodeghero. 2023. Under the Bridge: Trolling and the Challenges of Recruiting Software Developers for Empirical Research Studies. In *ICSE-NIER '23*. Association for Computing Machinery, Melbourne, Australia, 55–59. https://doi.org/10.1109/ICSE-NIER58687.2023.00016

[15] Brandon Kramer, Gizem Korkmaz, J Bayoán Santiago Calderon, and Carol Robbins. [n. d.]. International Collaboration in Open Source Software: A Longitudinal Network Analysis of GitHub. ([n. d.]).

[16] 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. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 9459–9474. https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf

[17] Arghavan Moradi Dakhel, 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

[18] Goutam Mylavarapu, Johnson P Thomas, and K Ashwin Viswanathan. 2019. An Automated Big Data Accuracy Assessment Tool. In *2019 IEEE 4th International Conference on Big Data Analytics (ICBDA)*. 193–197. https://doi.org/10.1109/ICBDA.2019.8713218

[19] Stack Overflow. [n. d.]. Stack Overflow Developer Survey 2023. https://survey.stackoverflow.co/2023/#ai. (Accessed on 04/04/2024).

[20] Pandoc. [n. d.]. Pandoc: a universal document converter. https://pandoc.org/. (Accessed on 05/14/2024).

[21] Gustavo Pinto, Isadora Cardoso-Pereira, Danilo Monteiro, Danilo Lucena, Alberto Souza, and Kiev Gama. 2023. Large Language Models for Education: Grading Open-Ended Questions Using ChatGPT. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering* (Campo Grande, Brazil) *(SBES '23)*. Association for Computing Machinery, New York, NY, USA, 293–302. https://doi.org/10.1145/3613372.3614197

[22] Rohith Pudari and Neil A Ernst. 2023. From copilot to pilot: Towards AI supported software development. *arXiv preprint arXiv:2303.04142* (2023).

[23] Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael 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* (Sydney, NSW, Australia) *(IUI '23)*. Association for Computing Machinery, New York, NY, USA, 491–514. https://doi.org/10.1145/3581641.3584037

[24] June Sallou, Thomas Durieux, and Annibale Panichella. 2023. Breaking the silence: the threats of using llms in software engineering. *arXiv preprint arXiv:2312.08055* (2023).

[25] Jaakko Sauvola, Sasu Tarkoma, Mika Klemettinen, Jukka Riekki, and David Doermann. 2024. Future of software development with generative AI. *Automated Software Engineering* 31, 1 (2024), 26. https://doi.org/10.1007/s10515-024-00426-z

[26] Igor Steinmacher, Sogol Balali, Bianca Trinkenreich, Mariam Guizani, Daniel Izquierdo-Cortazar, Griselda G Cuevas Zambrano, Marco Aurelio Gerosa, and Anita Sarma. 2021. Being a mentor in open source projects. *Journal of Internet Services and Applications* 12 (2021), 1–33. https://doi.org/10.1186/s13174-021-00140-z

[27] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. 2015. Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing* (Vancouver, BC, Canada) *(CSCW '15)*. Association for Computing Machinery, New York, NY, USA, 1379–1392. https://doi.org/10.1145/2675133.2675215

[28] Igor Steinmacher, Marco Aurélio Graciotto Silva, and Marco Aurélio Gerosa. 2014. Barriers Faced by Newcomers to Open Source Projects: A Systematic Review. In *Open Source Software: Mobile Open Source Technologies*, Luis Corral, Alberto Sillitti, Giancarlo Succi, Jelena Vlasenko, and Anthony I. Wasserman (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 153–163.

[29] Jan Stenberg. 2014. Exploring the Hexagonal Architecture. *InfoQ (Oct. 31, 2014). url: https://www. infoq. com/news/2014/10/exploring-hexagonal-architecture/(visited on 08/12/2019)* (2014).

[30] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* (2023).

[31] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Let's talk about it: evaluating contributions through discussion in GitHub. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Hong Kong, China) *(FSE 2014)*. Association for Computing Machinery, New York, NY, USA, 144–154. https://doi.org/10.1145/2635868.2635882

[32] S. Wagner, D. Mendez, M. Felderer, D. Graziotin, and M. Kalinowski. 2020. *Challenges in Survey Research*. Springer International Publishing, Cham, 93–125. https://doi.org/10.1007/978-3-030-32489-6_4

[33] Wei Wang, Huilong Ning, Gaowei Zhang, Libo Liu, and Yi Wang. 2024. Rocks Coding, Not Development–A Human-Centric, Experimental Evaluation of LLM-Supported SE Tasks. *arXiv preprint arXiv:2402.05650* (2024).