LLM-Based Chatbots for Mining Software Repositories: Challenges and Opportunities

Samuel Abedu
Concordia University
Montreal, Canada
samuel.abedu@mail.concordia.ca

Ahmad Abdellatif
University of Calgary
Calgary, Canada
ahmad.abdellatif@ucalgary.ca

Emad Shihab
Concordia University
Montreal, Canada
emad.shihab@concordia.ca

ABSTRACT

Software repositories have a plethora of information about software development, encompassing details such as code contributions, bug reports, code reviews, and project documentation. This rich source of data can be harnessed to enhance not only software quality and development velocity but also to gain insights into team collaboration, identify potential bottlenecks, and inform strategic decision-making throughout the software development lifecycle. Previous studies show that many stakeholders cannot benefit from the project information due to the technical knowledge and expertise required to extract the project data.

To lower the barrier to entry by automating the process of extracting and analyzing repository data, we explored the potential of using a large language model (LLM) to develop a chatbot for answering questions related to software repositories. We evaluated the chatbot on a set of 150 software repository-related questions. We found that the chatbot correctly answered one question about the repository. This result prompted us to shift our focus to investigate the challenges in adopting LLMs for the out-of-the-box development of software repository chatbots. We identified five main challenges related to retrieving data, structuring the data, and generating the answer to the user's query. Among these challenges, the most frequent (83.3%) is the inaccurate retrieval of data to answer questions. In this paper, we share our experience and challenges in developing an LLM-based chatbot to answer software repository-related questions within the SE community. We also provide recommendations on mitigating these challenges. Our findings will serve as a foundation to drive future research aimed at enhancing LLMs for adoption in extracting useful information from software repositories, fostering advancements in natural language understanding, data retrieval, and response generation within the context of software repository-related questions and analytics.

KEYWORDS

Software Chatbots, Large Language Model, Conversational Development Assistant

EASE 2024, 18–21 June, 2024, Salerno, Italy

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Samuel Abedu, Ahmad Abdellatif, and Emad Shihab. 2024. LLM-Based Chatbots for Mining Software Repositories: Challenges and Opportunities. In *Proceedings of The 28th International Conference on Evaluation and Assessment in Software Engineering (EASE 2024).* ACM, New York, NY, USA, 10 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Software repositories play a fundamental role in modern software development, containing a plethora of diverse information vital to the software development process. This includes source code, commit histories, bug reports, and documentation. The richness of data within these repositories offers a valuable opportunity for analysis, allowing for the derivation of actionable insights aimed at enhancing software quality and gaining a deeper understanding of the intricacies of software development processes [43, 19]. Researchers have extensively explored repository data to uncover various aspects of software evolution [8, 34], contribute to improvements in software quality [5, 46], establish requirement traceability [30], and analyze patterns in developer collaboration [36].

Extracting data from software repositories requires technical knowledge and expertise. For example, in a scenario where a stakeholder wants to identify the commit that caused a particular bug in a project. To achieve this, the stakeholder must first clone the code and issue repositories. Subsequently, write a script to extract and link data from these different sources to obtain the results. This process poses a significant barrier for non-technical stakeholders, including project managers and product owners [2]. Even if a stakeholder possesses the necessary skills to undertake such a task, it remains a time-consuming task. This complexity and technical nature of data extraction from repositories can impede the involvement of non-technical stakeholders in extracting valuable insights from the software development process.

Recently, there has been a significant advancement in the utilization of LLMs for various applications in software engineering[47, 13]. However, LLMs are hindered by their inability to access real-time information for knowledge-intensive tasks. Retrieval Augmented Generation (RAG) has been developed to make LLMs improve on knowledge-intensive tasks and generate factual up-to-date responses to queries [24]. RAG retrieves relevant information from an indexed data store to generate responses to queries. The RAG approach has been adopted by prior SE studies to improve code generation, code summarization and automatic program repair [50, 37, 28]. The success of the RAG approach on these SE tasks motivates us to propose an LLM-based chatbot to automate the process of extracting information from software repositories using the RAG approach. Given that software repository data is dynamic (i.e., updated frequently), using RAG ensures retrieving

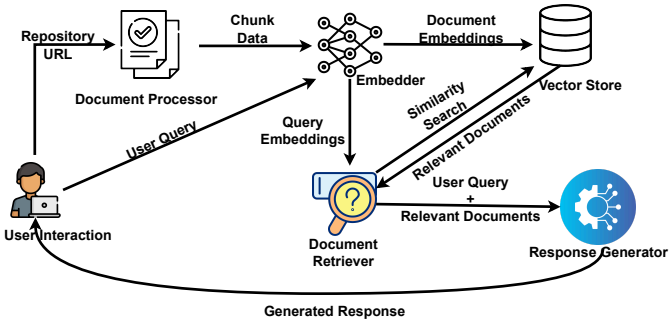

Figure 1: Overview of the Chatbot Components' Interaction.

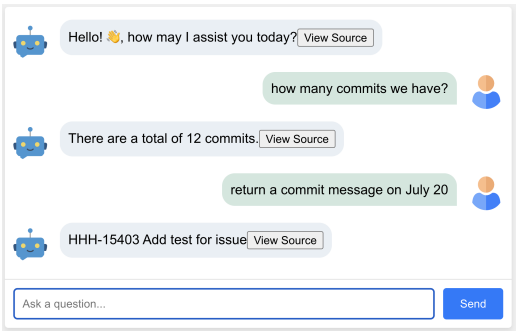

Figure 2: An example of a user conversation with the chatbot.

and providing the most recent updates in the repository to LLMs to generate a response based on the most recent data. Our RAG implementation of the chatbot has 5 main components namely, User interface, Document Processor, Embedder, Document Retriever and Response Generator.

We used a set of 150 repository-related questions [2] to evaluate the effectiveness of the chatbot. We find that out of the 150 questions posed to the chatbot, only one was answered correctly. Consequently, we shift our focus towards an in-depth investigation of the challenges associated with designing an LLM-based chatbot using RAG for extracting data from a software repository. To accomplish this, we conducted a manual analysis of the generated response and the output of each component to identify the challenges. This analysis serves as a crucial step in understanding the bottlenecks and refining the chatbot to enhance its efficacy in handling software repository-related queries. We believe that our findings have the potential to pave the way for future research to develop chatbots that provide non-technical stakeholders with insights derived from projects. The paper makes the following contributions:

- We share the challenges we encountered in developing the LLM chatbot. These are potential areas for improvement to make LLM more efficient in the field of SE.
- We make recommendations on possible ways to avoid or mitigate these challenges.
- We make the implementation and dataset publicly available to enhance the replication of our work[1].

Paper Organization. The paper is structured as follows: Section 2 covers our approach; Section 3 details implementation and use case; Section 4 discusses the challenges; Section 5 discusses findings; Section 6 addresses validity threats; Section 7 reviews related works; Section 8 concludes.

2 APPROACH

The initial goal of our study is to lower the barrier to entry for extracting useful information from software repositories by proposing an LLM-based chatbot to answer questions about software projects. Figure 1 presents the main components of our approach, namely (i) **User Interface** for user interaction with the chatbot, (ii) **Document Processor** that collects and preprocesses repository information, (iii) **Embedder**, which creates vector embeddings for the processed repository information for storage in a vector store, and vector embeddings for the user's query for semantic similarity search, (iv) **Document Retriever** that performs a semantic similarity search to get the relevant document to answer the user's question, and (v) **Response Generator** that generates responses from the selected documents using LLM. In this section, we describe each component in detail.

2.1 User Interface

To facilitate the extraction of software repository data, we introduce the User Interface (UI) component, which enables the user to interact with the chatbot through natural language. Through the UI, the user inputs the uniform resource locator (URL) of the target software repository they want to interact with and it forwards the URL to the Document Processor component (discussed next). Furthermore, the UI enables the user to pose questions to the chatbot and view the chatbot's responses.

Figure 2 shows the UI component where a user queries the chatbot in natural language, asking about the number of commits in the target repository. After the chatbot answers the user's query, the user asks the chatbot to return a commit information (message) from a specific date. To provide the user with the information used to generate the response, the UI includes a *"view source"* button that presents the output of the Document Retriever component (e.g., commit hashes and messages).

2.2 Document Processor

Software repositories contain rich and diverse sets of data, which can be analyzed to discover security vulnerabilities [59], extract requirements [30], and drive actionable insights about the software project [16]. After the user enters the URL through the UI component, the Document Processor retrieves metadata related to the repository (e.g., description, branches), commit (e.g., commit message, modified files) and issues (e.g., issue title, body). Then, the Document Processor component converts the content of all retrieved information into JSON format documents.

LLMs have strict token limits, meaning that they can only process a fixed number of tokens in an input or output sequence. To overcome this constraint, the Document Processor component splits the documents into chunks with a few tokens. Specifically, the Document Processor splits the retrieved metadata (e.g., commit message)

[1]https://zenodo.org/records/10526260

into smaller documents, called chunks, to avoid exceeding the token limit when working with LLMs. Previous work shows that a fragment size of 350 tokens demonstrated good performance when utilizing LLMs [56, 39]. Therefore, we configured the Document Processor component to split the documents into chunks of 350 token size. In addition, we included an overlap of 20 tokens in each document chunk to maintain continuity of context. In other words, the overlap ensures that the context at the end of one chunk is carried over to the beginning of the next, enhancing the model's ability to understand and process information that spans across multiple chunks. Next, the Document Processor indexes all the documents to track them after embedding them through their index. After the Document Processor preprocesses and indexes all the collected information, it forwards the chunks to the next component (Embedder) to create vector embeddings for each chunk.

2.3 Embedder

Vector embeddings and cosine similarity methods have been extensively used in previous SE studies to identify similar questions to a query on QA platforms like Stack Overflow [54], similarity between code snippets [57, 10], and match app reviews to bug reports [15]. Embeddings are vectors within a vector space that represent entities such as words, sentences, documents, or other data. They capture the semantic meaning of these entities, encoding aspects like context and relationships with other entities. In the vector space, semantically similar vector embeddings are positioned closely together, facilitating efficient similarity search [33, 22].

The Embedder component utilizes a transformer-based embedding model to generate embeddings for all chunks. These embeddings are then stored in a vector store. It is crucial to highlight that the Embedder, employing the same embedding model, generates an embedding for the user's query. Consequently, it streamlines an efficient search by the Document Retriever component, seeking similar embeddings within the repository information (chunks) that demonstrate high similarity with the user's query. The next subsection details the Document Retriever component.

2.4 Document Retriever

The main goal of the Document Retriever is to search among the document embeddings stored in the Vector store that are similar to the user's query embedding. Specifically, the Document Retriever finds document embeddings in the approximate nearest neighbours of the query embeddings and performs the semantic similarity search using the cosine similarity [32] on the document embeddings within the approximate nearest neighbourhood of the query embeddings. This is a more efficient approach compared to performing the search on the entire vector space. The Document Retriever then returns the most relevant documents to a query (documents whose cosine similarity score to the query is closer to 1). The cosine similarity metric normalizes the length of vectors [32], making it suitable for our chatbot—similarity searches between queries and documents with varying lengths. We configured the Document Retriever to return the top@4 documents similar to the user's query. This decision ensures the most relevant information is presented to the Response Generator and also adheres to the token limitation of LLMs. The retrieved documents serve as context for the query in the Response Generator.

> "You are an AI assistant for a software repository QA task, use the following pieces of context to answer the question at the end. If you don't know the answer, just say that you don't know, don't try to make up an answer. Use three sentences maximum. Keep the answer as concise as possible.
> *Context:* ***{context}***
> *Question:* ***{question}***
> *Answer:* "

Figure 3: Prompt used in this study.

2.5 Response Generator

The goal of the Response Generator component is to generate a response to the user's query based on documents retrieved by the Document Retriever. To achieve this, the Response Generator prompts the LLM to generate the response. To construct the prompt, we adhere to OpenAI's guidelines and best practices for prompt engineering[2] (e.g., being specific and detailed as possible about the context, outcome and format). Figure 3 presents the prompts used in our study. The prompt comprises three main parts: 1) an instruction to guide the behavior of the LLM, 2) Context, containing the retrieved documents relevant to the user's query, and 3) the user's query. To prevent the LLM from hallucinating, we instructed it to respond with *"I don't know"* if the answer is not evident in the given context. Once the LLM generates the answer, the Response Generator component returns it to the UI component for presentation to the user.

3 CASE STUDY

In this section, we present the software repositories and questions we use in the evaluation and explain the experimental settings used to achieve this goal.

3.1 Implementation

Before delving into performance evaluation, we describe the implementation settings of our approach to facilitate its replication on other datasets. We use Python's LangChain version 0.0.216[3] to implement the chatbot. Furthermore, we use GitHub's PyDriller, a Python wrapper for the GitHub API [43], in the Document Processor component to enable the retrieval of all relevant data for answering the queries as discussed in Section 2.2, such as commits and issues. For the Embedder implementation, we leverage the *embedding-ada-002* model[4] from OpenAI to generate vector embeddings for both document chunks and user queries, storing them in the Chroma vector store[5]. Prior studies show that the *embedding-ada-002* embedding model outperforms other specialized embedding models (e.g., CodeBERT) in code and natural language tasks [10]. To generate the chatbot's responses to user queries, the Response Generator

[2]https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-openai-api
[3]https://python.langchain.com/docs/get_started/introduction
[4]https://platform.openai.com/docs/models/embeddings
[5]https://www.trychroma.com/

component relies on OpenAI's *GPT3.5-turbo* model[6]. For the implementation, we select the default configurations of the parameters of RAG in the LangChain framework to evaluate the RAG under default configurations and evaluate the performance an average user would encounter when deploying RAG over software repositories.

3.2 Dataset

Table 1: Statistics of Hibernate and Kafka Projects.

Repository	Stars	Forks	Commits	Issues
Hibernate-ORM	5496	3316	16779	6808
Kafka	25481	12861	11470	14124

To determine the efficiency of using the LLM chatbot to answer software repository-related questions, we perform our experiment using Hibernate-ORM and Kafka repositories. We selected these repositories because they are popular open-source projects (more than 5,000 stars) and have a rich history (each has more than 11,000 commits and 6,000 bugs). Table 1 presents the number of stars, forks, commits and issues of each repository at the time of collection on-July 30, 2023. Also, these projects have been studied by prior SE work [2, 44, 7].

For the questions that the chatbot needs to answer, we utilize the test set employed in prior work by Abdellatif et al. [2]. These questions were collected from 12 participants inquiring about software repository-related questions. In total, the participants asked 165 questions representing 10 distinct. The set contains questions that inquire about the same intent but in different syntax. For example, *"Which commits fixed the bug id?"* could be asked as *"What are the fixing commits for the bug id?"*. It is important to note that we excluded 15 questions because they were either irrelevant (e.g., *"What's your name?"*) or ambiguous (e.g., *"August 1 till August 2"*). Table 2 presents the intents, their descriptions, examples, and the distribution of questions corresponding to each intent used in our evaluation. From the table, it is observed that the intents have a similar distribution of questions. These intents encompass questions related to the code repository (e.g., *"give me all the commits for NetworkClientUtils file"* from the ***File Commits*** intent), issues (e.g., *"which files introduced most bugs"* from the ***Buggy Files*** intent), or a combination of both code and issues (e.g., *"Determine the buggy commits that happened on Dec 19, 2017"* from the ***Buggy Fix Commit*** intent), covering various aspects of software-related questions in our evaluation.

3.3 Evaluation

To evaluate the chatbot's capability to correctly answer software repository questions discussed in Section 3.2, we pose all 150 questions to the chatbot and record the output of the Document Retriever and the Response Generator. Then, we manually check the generated response of each question to determine if it has been answered correctly or not based on the oracle. The response is classified as correct if it matches the oracle and incorrect if the response deviates from the oracle or is *"I don't know"* as stated in the prompt. The evaluation is based on the end-to-end performance of the chatbot, thus if the answer to the query is not contained in the documents retrieved by the document retriever, it is also labelled as incorrect. In cases where the chatbot answers incorrectly, we manually examine the outputs of each component to identify the reason for returning an incorrect answer.

To identify the challenges, we adopted open card sorting and multiple validation steps as used by prior work [4]. Specifically, the first author applied open card sorting to extract themes from the responses with the retrieved documents for all 150 questions. The themes represent the challenges faced by the chatbot. To ensure the robustness of these themes and reduce bias, the second author independently reviewed these themes. In cases where there was a disagreement, both annotators discuss the cases to reach an agreement. Finally, we discuss the findings with the third author to eliminate bias from the annotators. It is important to note that this analysis aims to broaden our comprehension of the challenges facing practitioners when using LLM over software repositories, rather than seeking an absolute truth about the most accurate themes in our data.

4 CHALLENGES

The results show that the LLM-based chatbot correctly answered one question (*"return a commit message on July 20"*). In this section, we shed light on the challenges that hinder the chatbot from answering repository-related questions. We find challenges related to data retrieval, data structuring, and response generation. Table 3 presents the challenges, their definitions, and the percentage of occurrence of each challenge in the evaluated questions. It is important to note that there are instances where we identify multiple challenges in a single question. In the following subsections, we detail each of the challenges.

4.1 Challenges Relating to Data Retrieval

Retrieving correct and relevant data is crucial for the chatbot's accuracy. In other words, if the retrieved documents are incorrect or do not include the necessary information to answer the user's query, the chatbot's final response would also be incorrect. Thus, it negatively impacts the user satisfaction with the chatbot. We identified two challenges related to data retrieval:

Challenge I. Out-of-the-box embedding models are not always accurate in tasks related to repository information retrieval.

The Document Retriever component returns inaccurate documents for 83.3% of the queries. Upon closer examination, we find two main reasons behind this observation. First, the Document Retriever component returns documents that do not capture the unique identifier specified in the query. For example, when the question *"commit(s) that fixed the bug ticket KAFKA-7354?"* is asked, the Document Retriever component returns documents related to a different bug, *"KAFKA-5716"*, leading the Response Generator to respond with *"I don't know"*. The Document Retriever component uses an embedding model, not specifically trained for repository information retrieval, which might not distinguish the importance of repository entities (bug ID or commit hash) from other words in

[6]https://platform.openai.com/docs/models/gpt-3-5

Table 2: Intents used in the evaluation adopted from [1].

Intent	Definition	Example Question	Frequency (%)
Buggy Commit	Identify the bugs introduced by specific commits.	Which are the bugs introduced by commit *Commit Hash*?	11 (7.3%)
Buggy Commits By Date	List the buggy commit(s) within a specific timeframe.	What is the percentage of bug fixing commits that introduced bugs in/on *Date*?	12 (8.0%)
Buggy Files	Identify the most buggy files to refactor.	Which are the most bug-introducing files?	12 (8.0%)
Fix Commit	Identify the commit(s) that fix a specific bug.	Which commits fixed the *bug id*?	17 (11.3%)
Buggy Fix Commits	Identify the commits that fixed a bug but also introduced a new bug on a specific date.	What are the buggy/fixing commits that happened in/on *Date*?	16 (10.7%)
Count Commits By Dates	Identify the number of commits within a specific period.	What is the number of commits in/on *Date*?	29 (19.3%)
Experienced Dev Fix Bugs	Identify the developer(s) who have experience fixing bugs related to a specific file.	Which developer(s) fixes the most bugs related to *File Name*?	14 (9.3%)
Overloaded Dev	Identify the developer(s) with the highest number of unresolved bugs.	Which developer(s) have the most unfixed bugs?	13 (8.7%)
File Commits	Examine the details about the changes that occurred on a specific file.	What is/are the latest commit(s) to *File Name*?	15 (10.0%)
Commits By Date	Identify the commit information on a specific date.	What commits were submitted on *Date*?	11 (7.33%)

the context when assigning values to capture semantic meaning. Thus, the embedding model does not capture the contextual relevance of the key details like the Bug Id. In our example (*commit(s) that fixed the bug ticket KAFKA-7354?*), the model appears to assign higher values to terms like *"commit"* which had more occurrence in the document pertaining to *"KAFKA-5716"* than the document pertaining to *"KAFKA-7354"*. This led to a higher cosine similarity score for the document pertaining to KAFKA-5716, despite the query's explicit reference to KAFKA-7354. This makes the Document Retriever component return the information of KAFKA-5716.

Also, the SE domain is a specialized domain with specific terminologies used in a unique manner. For instance, in the SE domain, the term "bug" refers to an error in the code that requires fixing, whereas in other domains, it denotes an insect. The second challenge is related to using out-of-the-box models to embed SE terminologies. To assess the Document Retriever's capability in returning the same or similar relevant documents for semantically similar questions with different SE terms, we randomly selected five questions and changed terms such as "bugs" to "issues" and "changes" to "commits". We then analyzed the documents returned in each case. We observed that the Document Retriever treats these terms as distinct words, failing to recognize their semantic equivalence in the SE context. For instance, the queries *"which commit(s) fixed the **bug** ticket HHH-11965"* and *"which commit(s) fixed the **issue** ticket HHH-11965"* are similar. However, the Document Retriever returns different documents for each question. Specifically, for the query with **"bug"**, the Document retriever returned only commit information, while for the query with **"issue"**, the Document retriever returned both commit and issue information.

Although there are existing word embedding models fine-tuned on Stack Overflow and data from GitHub like commit messages and issue body [11, 49], there is the need for evaluating their potential in embedding creation since their evaluation has focused mainly in masked language model (MLM) tasks where masked tokens are predicted. Furthermore, there is a need for more approaches that prioritize and understand the synonyms of SE key terms.

Mitigating Strategy. Fine-tuning LLMs with techniques like few-shot tuning for specific domain tasks has been shown to improve the accuracy of models like flanT5 [31]. As a solution, we recommend exploring the potential of fine-tuning the LLM/foundation model (FM) for domain-specific tasks. In the case of this challenge, an FM could be fine-tuned for sentence embedding specific to the task of software repository information retrieval.

Challenge II. The constraint on the number of tokens of LLMs does not permit to provide complete data for response generation.

LLMs adhere to a strict token limit; for example, the BART model has a token limit of 1024 [23]. Due to this constraint, we configured the Document Retriever component to return the top@4 documents with the highest similarity score to a user's query, as discussed in Section 2.4. This limitation hampers the Document Retriever component from providing the response generation component with all the documents required to generate the answer. Specifically, the Document Retriever component returns incomplete data for response generation in 49.3% of queries. The questions that require analyzing more data are particularly prone to this challenge, as they often require fetching a substantial amount of data to provide answers. For instance, in the question *"Who fixed the most bugs in HibernateEntityManager.java?"*, the Document Retriever component needs to retrieve all fixed bug information in the Hibernate project related to `HibernateEntityManager.java`. Then, provide

Table 3: The identified challenges and their definitions.

Category	Challenge	Definition	Frequency (%)
Data Retrieval	Out-of-the-box embedding models are not always accurate in tasks related to repository information retrieval	This challenge describes the ability of the embedding model to create embeddings that efficiently captures the semantics of SE data.	83.3%
	The constraint on the number of tokens of LLMs does not permit to provide complete data for response generation	The challenge is about retrieving limited data for a data-intensive query like the analytical questions due to LLM token limitation.	49.3%
Data Structuring	Chunking artefacts into smaller sizes to fit LLM token constraints leads to loss of information and relationship.	This challenge involves preserving contextual information and relationships when dividing documents into smaller token chunks to fit LLM token constraints.	24.0%
	The ambiguity and lack of clarity in the repository data affect the chatbot's response	The challenge is about the LLM's inability to comprehend information because of the ambiguity and lack of clarity in the data.	18.0%
Response Generation	The chatbot occasionally responds incorrectly despite retrieving correct contextual information	The challenge is about the chatbot's inability to accurately answer questions based on the contextual documents provided.	17.3%

the retrieved information to the Response Generator component to generate an accurate answer. This process involves returning all commits that fixed a bug in `HibernateEntityManager.java`, along with the authors, for the Response Generator to analyze and formulate a response. However, passing all this information to the Response Generator may lead to a token constraint error. On the other hand, returning limited information may not be sufficient for the Response Generator to generate an accurate response. We believe this challenge goes beyond the SE community, and practitioners who leverage LLM to perform analytical tasks that require a vast amount of data will encounter this challenge.

Mitigating Strategy. A solution will be to implement progressive prompting [25]. When the Response Generator receives a query which requires the analysis of a large amount of data, the Document Retriever can make this data available on demand. Specifically, when the Response Generator receives such questions and the data passed to it appears incomplete, it can request the Document Retriever to request additional data.

4.2 Challenges Relating to Data Structuring

The structured nature of software repository data is fundamental to understanding the development process. Some questions in our dataset require triangulation of diverse data sources, such as commits and bug information, to answer the user's query (e.g., *which bugs were introduced as a result of* ***commit hash***). However, triangulating these data posed a challenge to the Document Retriever.

Challenge III. Chunking artefacts into smaller sizes to fit LLM token constraints leads to loss of information and relationship.

We reiterate that we divided the repository data into smaller chunks of 350 tokens to address the token constraints of the LLMs, as discussed in Section 2.2. In contrast to challenge II, where the top@4 documents passed to the Response Generator may be incomplete, this challenge pertains to chunking the same document (e.g., metadata from a single commit) to keep within the token limit. The challenge with this process is the disconnection of related information. For example, consider the chunking of data from a specific commit, where, the commit hash and author details are in one chunk, and information about the files modified in the commit in another chunk. The lack of a direct link between the author's details and the modified files is a challenge. When the Document Retriever extracts the author's information from one chunk, it struggles to associate this data with the corresponding modified file in another chunk. This issue resulted in the loss of contextual information in the documents retrieved, impacting the generation of responses for 24.0% of the questions posed to the chatbot. Recent works present methods to overcome this challenge of linking data in segments [53, 52]. Nevertheless, further investigation is required by the SE community to validate those approaches in the domain.

Mitigating Strategy. The goal of the document chunking is to address the token limit constraints of LLMs. As mentioned earlier, Li et al. [25] proposed progressive prompting to solve this issue. Also, an issue with the document chunking was maintaining the continuity of context. We recommend context carryover to resolve reference to previous chunks [53]. Specifically, design the chunks such that each chunk will have a summary of the previous chunk.

Challenge IV. The ambiguity and lack of clarity in the repository data affect the chatbot's response.

In evaluating our chatbot, we find that the Document Retriever fails to process the information in the repository data. The main reason for this challenge is the ambiguity in the repository information such as the commit message. For example, in the query *"which commit(s) fixed the bug ticket HHH-11965"*, the Document Retriever failed to return documents containing commits referencing the specified bug ID (*"HHH-11965"*). Although there is a commit that fixes the mentioned issue, the commit message (*"HHH-11965 - Added test case"*) does not reference it as fixing a bug. This makes

the Document Retriever not recognize it as a fixing commit for the issue.

Mitigating Strategy. We recommend practitioners preprocess the data and explicitly link the documents together. For example, use the SZZ [42] to link the bugs with their fixing commits and explicitly provide the LLM with the links to generate a response. Furthermore, provide the LLM with more contextual data in the prompt to extrapolate the context and answer the user's query. For instance, adding the diff of changes will assist the LLM to know the type of changes being made (e.g. fixing bugs or refactoring code).

4.3 Challenges Relating to Response Generation

Retrieving accurate data to pass to the response generator is fundamental to the success of the chatbot. This is because it ensures that the chatbot can provide factual and reliable responses to user queries, contributing to a positive user experience. However, generating accurate responses based on the provided context is equally essential. The outputs of the Response Generator revealed a challenge, highlighting an area of improvement.

Challenge V. The chatbot occasionally responds incorrectly despite retrieving correct contextual information.

Our manual analysis shows that the Response Generator component generates incorrect answers for 17.3% of the posed questions, even though the Document Retriever component passed the correct context to the Response Generator component. For example, in the question *"changes on JobBatch"*, the Document Retriever correctly extracted documents that included the modification of the `JobBatch.java` file, which was provided to the Response Generator as context. However, the Response Generator replied with *"There is no information provided about any changes specifically made to the JobBatch"*. Another example is *"How many commits do we have?"*, where the Response Generator returns *"There are a total of 12 commits"*. Although there were five commit hashes in the documents passed as context to the Response Generator, it failed to aggregate the commits from the provided contextual information. We believe that this issue is distinct from hallucination because it involves the Response Generator producing inaccurate or misleading answers without fabricating information. Unlike hallucination, where the model might generate responses based on non-existent details, in this context, the Response Generator has access to the details provided as a part of the prompt, as discussed in Section 2.5. This issue highlights that, although the LLM is a state-of-the-art technology, it may not be perfect. The LLM appears to struggle to comprehend the information it was given to deduce the answer to questions, leading the chatbot to respond incorrectly. Further investigations by the research community are required to identify the underlying reasons for this issue. A potential area is to explore the impact of using different prompting strategies (e.g., chain of thoughts, few-shot learners) on the LLM's generated output.

Mitigating Strategy. LLMs are known to be good few-shot learners [21]. Adapting the approach of Zhang et al. [58], we recommend storing questions which are answered correctly and using these stored questions with their context and answers as learning examples for the LLM. When a user provides a new query, an example with the same intent as the user's query will be retrieved and used as a learning example for the LLM. That way, the LLM learns what information to extract from the context to answer correctly.

Also, another solution will be to fine-tune the LLM on software repository data. However, fine-tuning LLMs is an expensive process considering the number of trainable parameters. Nonetheless, there are parameter-efficient fine-tuning (PEFT) techniques like Low Rank Adaption (LoRA) [17], which have been proposed to make the fine-tuning process less expensive.

5 DISCUSSION

LLMs are revolutionizing various areas in the SE domain such as automated program repairs [20] and code review [27, 48]. However, its utilization in analyzing software repository data has not become mainstream [12]. In this study, we use an LLM-based chatbot to extract and analyze repository information. Our findings suggest that practitioners and researchers should make careful considerations when adopting out-of-the-box transformer models and LLMs for analyzing repository data.

Adopting LLMs to automate analyzing and drawing insights from software repository data will be beneficial, but it also poses several challenges. Our findings show that transformer-based embedding models and LLMs used out-of-the-box struggle with the terminologies found in the SE domain. The standard training of LLMs and transformer-based embedding models on a general corpus does not cover the distinct semantics of words in SE. Furthermore, our results show that these models often miss the semantic meaning of these SE terms in a context. This calls for additional effort into making embedding models and LLMs more accurate for SE tasks by fine-tuning with SE-specific datasets or researching novel ways for their adoption.

Moreover, the ambiguity in repository data poses a significant challenge. For example, commit messages may sometimes be vague even to humans, lacking the informative details required for LLMs to make accurate associations with other information in the repository. This might lead to inaccurate responses, akin to challenges IV as presented in Section 4.2. This opens up a future avenue for research, where the chatbot can ask for further clarifications (using the retrieved data) from the user to help resolve the ambiguity.

In our study, we created a token overlap for the continuity of context, which has proven to be efficient for the application of LLMs in other domains [6]. However, it was inefficient when applied to software repositories. Our results indicate that applying LLMs to analyze repository data demands a distinctive approach, particularly due to the unique characteristics of software repositories. The complexities within code, diverse project structures, and the intricacies of version control systems (e.g., commit, PRs, issues) underscore the necessity for specialized strategies in effectively harnessing LLMs for comprehensive understanding.

The potential for applying LLMs in analyzing repository data is promising, with prospects of evolving the current manual ways of mining and analyzing repository data into an efficient, automated workflow. However, there are challenges that have been highlighted in this study that need to be addressed. Addressing these challenges will ensure that LLMs will not only comprehend the source code of repositories but will understand and analyze the

associated metadata of the repository that tells a story about the development process.

6 THREATS TO VALIDITY

In this section, we discuss the threats to our study's construct, internal, and external validity.

Construct Validity. A threat to construct validity in our study arises from instructing the LLM to respond with "I don't know" when it is unsure of the answer. LLMs sometimes hallucinate and we make the assumption that instructing the LLM to respond "I don't know" mitigates the risk of hallucinations. Nonetheless, the adopted RAG approach also reduces hallucinations [41].

Internal Validity. An internal threat to the validity of our study is the quality of the software repository data that our chatbot utilized. If the data collected is incomplete, it could lead to the chatbot giving an incorrect or partial answer. To eliminate this threat, we manually inspected the repository metadata to ensure that all the answers to the queries were present.

In our study, we examined the output of the Document Retriever and the Response Generator for each query to identify the issues that lead the chatbot to respond incorrectly. A threat to validity is the personal bias of the annotators during this process. To ensure that the bias was eliminated, the annotators discussed the issues identified for the questions to reach a consensus on the challenge present for the question.

External Validity. Threats to external validity concern the generalization of our findings. In the evaluation of our chatbot, we adopted the settings of Abdellatif et al. [2], meaning we evaluated the chatbot on two repositories, Hibernate-ORM and Kafka. A threat from this decision means our results and findings might not generalize to other software repositories. However, the Hibernate-ORM and Kafka are used by prior SE work and they are relatively large software repositories which makes it ideal for evaluating our chatbot rather than selecting relatively small repositories with fewer commits and bugs.

Also, we used the default parameters of the RAG implementation in the LangChain framework and selected chunk size and overlap based on prior studies. Although these values are selected based on prior studies, different configurations could potentially lead to different results, affecting the overall generalizability of our results. Our ability to experiment with different chunks and overlap sizes was constrained by the costs involved in running the experiments, thus limiting the generalizability of our results. Another threat to external validity is that we used OpenAI's *ada-text-embedding* and *gpt-3.5-turbo* models in this study. This means our findings might not generalize when utilizing other embedding models and large language models.

7 RELATED WORKS

In this section, we present the studies related to chatbots in the SE domain and discuss the work that leverages LLMs in the SE domain.

Software Engineering Chatbots. Chatbots have become popular in software engineering with a dedicated workshop on it [40]. Chatbots have been proposed to automate different software engineering tasks to enhance developers' and other stakeholders' experience. For instance, Abdellatif et al. [2] addresses the challenge of extracting repository information by implementing MSRBot. MSRBot simplified the process of extracting relevant information from software repositories by allowing the user to input a question/task in a natural language. Their findings indicated that most participants found MSRBot useful, completing tasks faster and more accurately compared to manual methods. Also, Xu et al. [54] implemented Answerbot, a chatbot which focuses on summarizing multi-answer posts from technical Q&A sites like Stack Overflow. AnswerBot effectively retrieves and summarizes relevant answers for developers, as validated by user studies. Again, Dominic et al. [9] implemented a chatbot to assist newcomers in contributing to open-source projects by recommending suitable projects and resources to them. Okanović et al. [35] introduces PerformoBot, a chatbot designed to guide developers in configuring and executing load tests, with positive feedback from users, particularly those less familiar with performance engineering. Yu et al. [55] introduces CodeMaster, an approach that employs CodeT5, for answering code questions. CodeMaster is described as state-of-the-art with its performance on the CodeQA benchmark by answering questions like What is the length of array X in a code sample. On the quality assurance of chatbots, López-Morales et al. [29] presents Asymob, a platform enabling chatbot quality measurement across different technologies, aiding in detecting quality issues in chatbots.

These studies show the diverse applications of chatbots in software engineering. The closest of these chatbots to our work is MSRBot [2]. However, the MSRBot supports a limited number of questions. Specifically, MSRBot only supports questions within their defined intents and questions that do not represent these intents are not supported. Nonetheless, the ability of LLMs to understand a variety of questions in a zero, one or few-shot setting has been demonstrated in a multiple choice question answering scenario [38]. With this, we project that using LLMs, we will be able to support a variety of questions compared to MSRBot [2].

LLMs in Software Engineering. Prior studies on using LLMs for software engineering tasks mainly focus on program repairs [13, 20], code generation [18, 47, 51], code reviews [26, 27, 48] and code summarization [3, 14, 45]. Fan et al. [13] studied the potential of Automated Program Repair (APR) techniques in rectifying errors in Codex-generated code for LeetCode contests. They highlight Codex's better performance over tools like TBar and Recoder. They recommend enhancing APR tools to overcome constraints in patch space since LLMs can generate more fixed patterns when trained on larger datasets. Kang et al. [20] proposed LIBRO, a framework that leverages LLMs to generate tests from bug reports with success in the Defects4J benchmark. Wang et al. [51] proposed "CodeT5+", an encoder-decoder LLM that supports a variety of downstream code tasks like natural language to code generation tasks, surpassing models like OpenAI's code-cushman-001. Jain et al. [18] developed "Jigsaw", a tool focusing on enhancing LLMs' semantic understanding of code. Tian et al. [47] evaluated ChatGPT as a programming assistant by assessing its capabilities in code generation, program repair and code summarization. Tufano et al. [48] fine-tuned a T5 model to automate code reviews. The fine-tuned T5 model was used to automate code-to-code tasks, code and comment-to-code tasks and code-to-comment tasks. Li et al. [26] introduced AUGER, a tool using the T5 model to automate code reviews. It achieves a

37.38% improvement in ROUGE-L. Similarly, Li et al. [27] proposed CodeReviewer, a pre-trained model tailored for code review. On code summarization, Geng et al. [14] discussed LLMs' potential in generating diverse code comments, surpassing traditional supervised learning approaches. Ahmed and Devanbu [3] evaluated GPT Codex on code summarization on project-specific data using few-shot learning. Also, Sun et al. [45] compared ChatGPT's performance against leading code summarization models like CodeBert and CodeT5. Their findings show that ChatGPT code summarisation performance is worse compared to the other models.

To the best of our knowledge, this is the first study employing LLMs to analyze software repository data beyond source code. We believe that our work paves the way for future research, exploring the use of LLMs in software repositories to provide technical insights from projects to non-technical stakeholders, akin to interacting with a chatbot.

8 CONCLUSION

Software repositories contain a vast amount of valuable data with the potential to enhance software projects. However, not all project stakeholders possess the technical expertise and time required to extract such data from repositories [2]. In this paper, we aim to explore the effectiveness of a chatbot powered by an LLM in answering questions related to software repositories. Therefore, we build a chatbot using the RAG approach to answer software repository-related questions. We evaluated the chatbot using a curated set of 150 questions. Our evaluation revealed that the chatbot accurately answered only one question, providing inaccurate answers for the remaining queries. These results highlight the limitations of deploying LLMs out-of-the-box for software engineering tasks. To delve deeper into the challenges associated with utilizing an LLM-powered chatbot for answering software repository questions, we manually examine the results. We identified five challenges related to retrieving data, structuring the data, and generating responses to user queries. The most recurring challenge pertains to using the embedding model out-of-the-box which affected the accurate retrieval of information to serve as context for the LLM. Additionally, the data chunking process impacted the chatbot's performance, and in some instances, we identified issues with the LLM's ability to answer questions based on the information provided as context. These findings underscore the complexities involved in implementing LLMs for specific software engineering tasks and point to the need for tailored solutions to enhance their performance in the context of software repositories.

We believe that our study provides valuable lessons for the community. Our findings will serve as a cautionary insight, informing practitioners to exercise caution when adopting LLMs for tasks involving SE-specific terminologies. The challenges observed in our study underscore the importance of understanding the context-specific nature of SE language when integrating LLMs into such applications. For researchers, our findings present an opportunity to open future research avenues in investigating ways to enhance the performance of LLMs for SE-specific tasks, particularly in the context of answering repository-related questions. Addressing the challenges identified in our study could lead to the development of more effective and specialized chatbots tailored to the specific needs of software engineering tasks. In future work, we will focus on evaluating the impact of different parameters on the results. Specifically, we will evaluate the impact of different chunk sizes and overlaps on the results

REFERENCES

[1] Ahmad Abdellatif, Khaled Badran, Diego Elias Costa, and Emad Shihab. 2022. A Comparison of Natural Language Understanding Platforms for Chatbots in Software Engineering. *IEEE Transactions on Software Engineering*, 48, 8, (Aug. 2022), 3087–3102.

[2] Ahmad Abdellatif, Khaled Badran, and Emad Shihab. 2020. MSRBot: Using bots to answer questions from software repositories. *Empirical Software Engineering*, 25, 3, (May 2020), 1834–1863.

[3] Toufique Ahmed and Premkumar Devanbu. 2022. Few-shot training LLMs for project-specific code-summarization. (Sept. 2022). arXiv: 2207.04237 [cs].

[4] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software Engineering for Machine Learning: A Case Study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. (May 2019), 291–300.

[5] Justus Bogner and Manuel Merkel. 2022. To type or not to type? a systematic comparison of the software quality of JavaScript and typescript applications on GitHub. In *Proceedings of the 19th International Conference on Mining Software Repositories* (MSR '22). Association for Computing Machinery, New York, NY, USA, (Oct. 2022), 658–669.

[6] Sebastian Borgeaud et al. 2022. Improving Language Models by Retrieving from Trillions of Tokens. In *Proceedings of the 39th International Conference on Machine Learning*. PMLR, (June 2022), 2206–2240.

[7] Aleksandr Chueshev, Julia Lawall, Reda Bendraou, and Tewfik Ziadi. 2020. Expanding the Number of Reviewers in Open-Source Projects by Recommending Appropriate Developers. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. (Sept. 2020), 499–510.

[8] Malinda Dilhara, Ameya Ketkar, and Danny Dig. 2021. Understanding Software-2.0: A Study of Machine Learning Library Usage and Evolution. *ACM Transactions on Software Engineering and Methodology*, 30, 4, (July 2021), 1–42.

[9] James Dominic, Jada Houser, Igor Steinmacher, Charles Ritter, and Paige Rodeghero. 2020. Conversational Bot for Newcomers Onboarding to Open Source Projects. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops* (ICSEW'20). Association for Computing Machinery, New York, NY, USA, (Sept. 2020), 46–50.

[10] Shihan Dou et al. 2023. Towards Understanding the Capability of Large Language Models on Code Clone Detection: A Survey. (Aug. 2023). arXiv: 2308.01191 [cs].

[11] Vasiliki Efstathiou, Christos Chatzilenas, and Diomidis Spinellis. 2018. Word Embeddings for the Software Engineering Domain. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. (May 2018), 38–41.

[12] 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. (Oct. 2023). arXiv: 2310.03533 [cs].

[13] Zhiyu Fan, Xiang Gao, Martin Mirchev, Abhik Roychoudhury, and Shin Hwei Tan. 2023. Automated Repair of Programs from Large Language Models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. (May 2023), 1469–1481.

[14] Mingyang Geng, Shangwen Wang, Dezun Dong, Haotian Wang, Ge Li, Zhi Jin, Xiaoguang Mao, and Xiangke Liao. 2023. Large Language Models are Few-Shot Summarizers: Multi-Intent Comment Generation via In-Context Learning. (June 2023). arXiv: 2304.11384 [cs].

[15] Marlo Haering, Christoph Stanik, and Walid Maalej. 2021. Automatically Matching Bug Reports With Related App Reviews. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. (May 2021), 970–981.

[16] Ahmed E. Hassan. 2008. The road ahead for Mining Software Repositories. In *2008 Frontiers of Software Maintenance*. (Sept. 2008), 48–57.

[17] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. (Oct. 2021). arXiv: 2106.09685 [cs].

[18] Naman Jain, Skanda Vaidyanath, Arun Iyer, Nagarajan Natarajan, Suresh Parthasarathy, Sriram Rajamani, and Rahul Sharma. 2022. Jigsaw: large language models meet program synthesis. In *Proceedings of the 44th International Conference on Software Engineering* (ICSE '22). Association for Computing Machinery, New York, NY, USA, (July 2022), 1219–1231.

[19] Yasutaka Kamei and Andy Zaidman. 2020. Guest editorial: Mining software repositories 2018. *Empirical Software Engineering*, 25, 3, (May 2020), 2055–2057.

[20] Sungmin Kang, Juyeon Yoon, and Shin Yoo. 2023. Large language models are few-shot testers: exploring llm-based general bug reproduction. In *Proceedings*

of the 45th International Conference on Software Engineering (ICSE '23). IEEE Press, Melbourne, Victoria, Australia, 2312–2323.

[21] Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large Language Models are Zero-Shot Reasoners. *Advances in Neural Information Processing Systems*, 35, (Dec. 2022), 22199–22213.

[22] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32* (ICML'14). JMLR.org, Beijing, China.

[23] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. (Oct. 2019). arXiv: 1910.13461 [cs, stat].

[24] Patrick Lewis et al. 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. (Apr. 2021). arXiv: 2005.11401 [cs].

[25] Haonan Li, Yu Hao, Yizhuo Zhai, and Zhiyun Qian. 2023. The Hitchhiker's Guide to Program Analysis: A Journey with Large Language Models. (July 2023). arXiv: 2308.00245 [cs].

[26] Lingwei Li, Li Yang, Huaxi Jiang, Jun Yan, Tiejian Luo, Zihan Hua, Geng Liang, and Chun Zuo. 2022. AUGER: automatically generating review comments with pre-training models. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (ESEC/FSE 2022). Association for Computing Machinery, New York, NY, USA, (Nov. 2022), 1009–1021.

[27] Zhiyu Li et al. 2022. Automating code review activities by large-scale pre-training. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (ESEC/FSE 2022). Association for Computing Machinery, New York, NY, USA, (Nov. 2022), 1035–1047.

[28] Shangqing Liu, Yu Chen, Xiaofei Xie, Jingkai Siow, and Yang Liu. 2021. Retrieval-Augmented Generation for Code Summarization via Hybrid GNN. (May 2021). arXiv: 2006.05405 [cs].

[29] Jose María López-Morales, Pablo C. Cañizares, Sara Pérez-Soler, Esther Guerra, and Juan de Lara. 2022. Asymob: a platform for measuring and clustering chatbots. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings* (ICSE '22). Association for Computing Machinery, New York, NY, USA, (Oct. 2022), 16–20.

[30] Yijing Lyu, Heetae Cho, Pilsu Jung, and Seonah Lee. 2023. A Systematic Literature Review of Issue-Based Requirement Traceability. *IEEE Access*, 11, 13334–13348.

[31] Zeyang Ma, An Ran Chen, Dong Jae Kim, Tse-Hsun Chen, and Shaowei Wang. 2024. LLMParser: an exploratory study on using large language models for log parsing. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (ICSE '24). ACM, New York, NY, USA.

[32] Christopher Manning, Prabhakar Raghavan, and Hinrich Schuetze. 2009. Introduction to Information Retrieval.

[33] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. (Sept. 2013). arXiv: 1301.3781 [cs].

[34] Lloyd Montgomery, Clara Lüders, and Walid Maalej. 2022. An alternative issue tracking dataset of public Jira repositories. In *Proceedings of the 19th International Conference on Mining Software Repositories* (MSR '22). Association for Computing Machinery, New York, NY, USA, (Oct. 2022), 73–77.

[35] Dušan Okanović, Samuel Beck, Lasse Merz, Christoph Zorn, Leonel Merino, André van Hoorn, and Fabian Beck. 2020. Can a Chatbot Support Software Engineers with Load Testing? Approach and Experiences. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering* (ICPE '20). Association for Computing Machinery, New York, NY, USA, (Apr. 2020), 120–129.

[36] Gabriel P. Oliveira, Ana Flávia C. Moura, Natércia A. Batista, Michele A. Brandão, Andre Hora, and Mirella M. Moro. 2023. How do developers collaborate? Investigating GitHub heterogeneous networks. *Software Quality Journal*, 31, 1, (Mar. 2023), 211–241.

[37] Md Rizwan Parvez, Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Retrieval Augmented Code Generation and Summarization. (Sept. 2021). arXiv: 2108.11601 [cs].

[38] Joshua Robinson, Christopher Michael Rytting, and David Wingate. 2023. Leveraging Large Language Models for Multiple Choice Question Answering. (Mar. 2023). arXiv: 2210.12353 [cs].

[39] Junxiao Shen, John Dudley, and Per Ola Kristensson. 2023. Encode-Store-Retrieve: Enhancing Memory Augmentation through Language-Encoded Egocentric Perception. (Aug. 2023). arXiv: 2308.05822 [cs].

[40] Emad Shihab, Stefan Wagner, and Marco Aurélio Gerosa. 2021. Summary of the 2nd International Workshop on Bots in Software Engineering (BotSE 2020). *ACM SIGSOFT Software Engineering Notes*, 46, 1, (Feb. 2021), 20–22.

[41] Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. 2021. Retrieval Augmentation Reduces Hallucination in Conversation. (Apr. 2021). arXiv: 2104.07567 [cs].

[42] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. 2005. When do changes induce fixes? *ACM SIGSOFT Software Engineering Notes*, 30, 4, (May 2005), 1–5.

[43] Davide Spadini, Maurício Aniche, and Alberto Bacchelli. 2018. PyDriller: Python framework for mining software repositories. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (ESEC/FSE 2018). Association for Computing Machinery, New York, NY, USA, (Oct. 2018), 908–911.

[44] Murali Sridharan, Mika Mäntylä, Maëlick Claes, and Leevi Rantala. 2022. SoCCMiner: a source code-comments and comment-context miner. In *Proceedings of the 19th International Conference on Mining Software Repositories* (MSR '22). Association for Computing Machinery, New York, NY, USA, (Oct. 2022), 242–246.

[45] Weisong Sun et al. 2023. Automatic Code Summarization via ChatGPT: How Far Are We? (May 2023). arXiv: 2305.12865 [cs].

[46] Patanamon Thongtanunam and Ahmed E. Hassan. 2021. Review Dynamics and Their Impact on Software Quality. *IEEE Transactions on Software Engineering*, 47, 12, (Dec. 2021), 2698–2712.

[47] Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F. Bissyandé. 2023. Is ChatGPT the Ultimate Programming Assistant – How far is it? (Apr. 2023). arXiv: 2304.11938 [cs].

[48] Rosalia Tufano, Simone Masiero, Antonio Mastropaolo, Luca Pascarella, Denys Poshyvanyk, and Gabriele Bavota. 2022. Using pre-trained models to boost code review automation. In *Proceedings of the 44th International Conference on Software Engineering* (ICSE '22). Association for Computing Machinery, New York, NY, USA, (July 2022), 2291–2302.

[49] Julian von der Mosel, Alexander Trautsch, and Steffen Herbold. 2023. On the Validity of Pre-Trained Transformers for Natural Language Processing in the Software Engineering Domain. *IEEE Transactions on Software Engineering*, 49, 4, (Apr. 2023), 1487–1507.

[50] Weishi Wang, Yue Wang, Shafiq Joty, and Steven C.H. Hoi. 2023. RAP-Gen: Retrieval-Augmented Patch Generation with CodeT5 for Automatic Program Repair. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (ESEC/FSE 2023). Association for Computing Machinery, New York, NY, USA, (Nov. 2023), 146–158.

[51] Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi D. Q. Bui, Junnan Li, and Steven C. H. Hoi. 2023. CodeT5+: Open Code Large Language Models for Code Understanding and Generation. (May 2023). arXiv: 2305.07922 [cs].

[52] Kai Wei et al. 2021. Attentive Contextual Carryover for Multi-Turn End-to-End Spoken Language Understanding. In *2021 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. (Dec. 2021), 837–844.

[53] Chunyang Wu, Yongqiang Wang, Yangyang Shi, Ching-Feng Yeh, and Frank Zhang. 2020. Streaming Transformer-based Acoustic Models Using Self-attention with Augmented Memory. (May 2020). arXiv: 2005.08042 [cs, eess].

[54] Bowen Xu, Zhenchang Xing, Xin Xia, and David Lo. 2017. AnswerBot: Automated generation of answer summary to developers' technical questions. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. (Oct. 2017), 706–716.

[55] Tingrui Yu, Xiaodong Gu, and Beijun Shen. 2022. Code Question Answering via Task-Adaptive Sequence-to-Sequence Pre-training. In *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*. (Dec. 2022), 229–238.

[56] Yuheng Zha, Yichi Yang, Ruichen Li, and Zhiting Hu. 2023. AlignScore: Evaluating Factual Consistency with a Unified Alignment Function. (May 2023). arXiv: 2305.16739 [cs].

[57] Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation. (Oct. 2023). arXiv: 2303.12570 [cs].

[58] Jieyu Zhang, Ranjay Krishna, Ahmed H. Awadallah, and Chi Wang. 2023. EcoAssistant: Using LLM Assistant More Affordably and Accurately. (Oct. 2023). arXiv: 2310.03046 [cs].

[59] Jiayuan Zhou, Michael Pacheco, Zhiyuan Wan, Xin Xia, David Lo, Yuan Wang, and Ahmed E. Hassan. 2021. Finding A Needle in a Haystack: Automated Mining of Silent Vulnerability Fixes. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. (Nov. 2021), 705–716.