RepoChat: An LLM-Powered Chatbot for GitHub Repository Question-Answering

Samuel Abedu*, Laurine Menneron†, SayedHassan Khatoonabadi*, Emad Shihab*
*Department of Computer Science and Software Engineering
Concordia University, Montreal, Canada
Email: samuel.abedu@mail.concordia.ca, {sayedhassan.khatoonabadi, emad.shihab}@concordia.ca
†CESI Graduate School of Engineering, Saint-Nazaire, France
Email: laurine.menneron@viacesi.fr

***Abstract*—Software repositories contain a wealth of data about the software development process, such as source code, documentation, issue tracking, and commit histories. However, accessing and extracting meaningful insights from these data is time-consuming and requires technical expertise, posing challenges for software practitioners, especially non-technical stakeholders like project managers. Existing solutions, such as software engineering chatbots leveraging LLMs, have demonstrated significant limitations in retrieving relevant data to answer user questions. In this paper, we introduce RepoChat, a web-based tool designed to answer repository-related questions by synergizing LLMs with knowledge graphs. RepoChat operates in two steps: (1) the Data Ingestion step, where it collects and constructs a knowledge graph from repository metadata, such as commits, issues, files and users; and (2) the Interaction step, where it takes the users natural language question, translates it into graph queries using an LLM, executes these queries against the knowledge graph, and generates a user-friendly response to the question using the query results as context. We evaluate RepoChat by conducting a user study in which participants asked a series of repository-related questions representing common developer intents. RepoChat achieved an accuracy of 90%, correctly answering 36 out of 40 questions, demonstrating its effectiveness in accurately retrieving relevant information to answer user's questions. RepoChat is available at https://repochattool.streamlit.app/, and its source code is accessible on Zenodo [1].**

***Keywords*-**Mining Software Repositories, Software Engineering Chatbots, Software Development Assistants

I. Introduction

Software repositories contain a plethora of information essential to the software development process. Data such as source code, documentation, issue tracking records, and commit histories [2] offer valuable insights aimed at improving software quality when harnessed effectively [3]. Software practitioners are interested in analyzing specific questions about their projects that require mining and interpreting repository data [4], [5]. However, accessing and extracting meaningful insights from repositories is often time-consuming and demands significant technical expertise [6], [7].

Prior studies have addressed this challenge by developing a chatbot that provides intuitive natural language interfaces to software repositories [6], [8]. For instance, Abdellatif et al. [6], proposed a chatbot relying on the natural language understanding (NLU) approach to answer repository-related questions. A challenge with the NLU approach in chatbot development is accurately interpreting user queries and mapping them to appropriate data retrieval actions [9]. Also, intents in the NLU approach are mapped to specific predefined actions [10]. Recent progress in large language models (LLMs) has demonstrated improved capabilities in understanding natural language queries and identifying user intents [11]. However, leveraging LLMs to build chatbots for repository question answering (QA) has proven challenging. Abedu et al. [8] found that LLM-based chatbots using the retrieval augmented generation (RAG) approach often fail to retrieve accurate data for repository-related questions, which was observed in 83.3% in their evaluated questions.

To address these challenges, we introduce **RepoChat**, a web-based tool for answering repository-related questions based on our approach in [12]. RepoChat synergizes LLMs and knowledge graphs to enhance the accuracy of LLM-based chatbots for software repository question-answering. RepoChat operates in two steps: Data Ingestion and Interaction. In the Data Ingestion step, the tool constructs a knowledge graph from repository data by collecting information such as commits, issues, and files. The knowledge graph models the complex relationships inherent in software repositories and store in a graph database for querying. In the Interaction step, RepoChat translates natural language questions into graph queries, executes these queries against the knowledge graph, and generates coherent, user-friendly answers.

To evaluate RepoChat, we defined a set of 10 tasks representing the intent of questions commonly asked by software practitioners [13] and conducted a user study to assess the tool's performance. Our evaluation shows that RepoChat achieves a 90% accuracy in providing correct answers to repository-related questions. We make available the source code for RepoChat at [1] to facilitate reproducibility and advance the field. RepoChat is also accessible online at https://repochattool.streamlit.app.

The remainder of the paper is organized as follows: Section II details the design of RepoChat, including its Data Ingestion and interaction steps. Section III describes the technologies used for the implementation of the tool. Section IV presents the evaluation methodology and results. Section V reviews related work, and Section VI concludes the paper and discusses future work.

II. Tool Design

RepoChat is a web-based application that enables users to interact with the metadata of their software repositories using natural language. Figure 1 overviews the architecture of RepoChat. The tool implements the approach presented in [12]. The design follows a two-step approach: (1) the Data Ingestion step, where the knowledge graph is constructed for the repository data, and (2) the Interaction step, where the user provides the question and gets a natural language response.

In the Data Ingestion step, RepoChat constructs a knowledge graph from the repository's metadata. It first collects the relevant data such as commit history, issues, pull requests, files and collaborator information of the project from GitHub. After the data collection, it identifies the bug-fixing commits by searching for the bug ID in the change log of the commits and then uses the SZZ algorithm [14] to identify the changes that introduced the bugs before constructing the knowledge graph. The knowledge graph is a structured representation of entities within the repository. It includes entities such as commits, issues, files, pull requests and users and their relationships based on the official GitHub schema [1]. By leveraging knowledge graph, the tool models complex interconnections within the repository, which facilitates analysis and inference [15], enabling the tool to answer complex repository-related questions. The knowledge graph follows established practices in knowledge representation by capturing multi-relational data in the form of triple facts (*head entity*, `relationship`, *tail entity*) [16]

In the Interaction step, RepoChat translates user questions into graph queries to retrieve the relevant information from the knowledge graph and generate a user-friendly response. The Interaction step has three components: Query Generator, Query Executor and Response Generator. The Query Generator component utilizes an LLM to generate a graph query corresponding to the user's natural language question. The prompt for the LLM in the Query Generator incorporates prompt engineering best practices and guidelines [17] to guide the LLM produce accurate and efficient queries that align with the knowledge graph's schema. Once the graph query is generated, the Query Executor extracts the graph query from the text generated by the Query Generator and executes it against the stored knowledge graph, retrieving relevant data to answer the user's question. The Response Generator then interprets the results and generates a natural language response to the user's question.

III. Implementation

RepoChat is implemented as a web application with a backend API consisting of two endpoints—the Data Ingestion endpoint and the chat endpoint—and a frontend interface that allows users to interact with it. The implementation is done with the Langchain framework, a framework that facilitates the building of LLM applications [2] and Streamlit, a Python framework for frontend development [3]. Figure 2 demonstrates the workflow of the data ingestion process, and Figure 3 shows the workflow of the interaction process.

A. Backend

In the backend, the Data Ingestion endpoint collects data from GitHub and constructs the knowledge graph, and the Chat endpoint interacts with repository data.

1) Ingest endpoint: The ingest endpoint accepts user inputs which include the repository URL, GitHub personal access token with permissions to query the GitHub GraphQL API[4], and the graph database credentials (URI, username, and password).

The Data Ingestion process begins by cloning the specified repository and extracting commit information using Git commands executed in a Z shell (zsh). It then utilizes the GitHub GraphQL API to collect detailed repository metadata, including descriptions, issues, and collaborators. To identify bug-fixing and bug-introducing commits, it searches for the Bug ID in the change log of the commits to identify the bug-fixing commits and uses the R-SZZ variant of the SZZ algorithm to identify the bug-introducing commits [18]. After collecting the data, it constructs the knowledge graph by mapping the entities and relationships as described in Section II. The constructed knowledge graph is then stored in a Neo4j database. Upon completion of the ingestion process, the endpoint returns a status indicating the success or failure of the process.

2) Chat endpoint: The chat endpoint facilitates the Interaction step, handling the user's question and generating the relevant response. The chat endpoint accepts as input the user's question and OpenAI's API key. It uses the GPT-4o model to generate the Cypher query for the user's question and also generate a final response from the retrieved data. The temperature of the GPT-4o for generating the Cypher query is set to zero to reduce the randomness in the LLMs generation [19]. That for generating the final response is set to the default temperature of 0.7. Before generating the Cypher query, the conversation history is retrieved from a Redis cache and added to the prompt to give the LLM additional context about the user's previous questions and interactions. The question is incorporated into a Cypher prompt template, which includes the schema of the knowledge graph to guide the language model in generating a corresponding Cypher query. The prompt follows best practices in prompt engineering [17], ensuring that the language model produces accurate and relevant queries. Once the language model generates the Cypher query, it is parsed through a regular expression matching to extract only the Cypher query. This step addresses potential issues where the model includes additional text alongside the query, which could lead to execution errors. The extracted Cypher query is then executed against the Neo4j database using the Neo4j Python driver. The results retrieved from the database are passed to a response generation model, which constructs a natural language answer to the user's question.

[1] https://docs.github.com/en/graphql/overview/public-schema

[2] https://www.langchain.com/

[3] https://streamlit.io/

[4] https://docs.github.com/en/graphql/guides/forming-calls-with-graphql

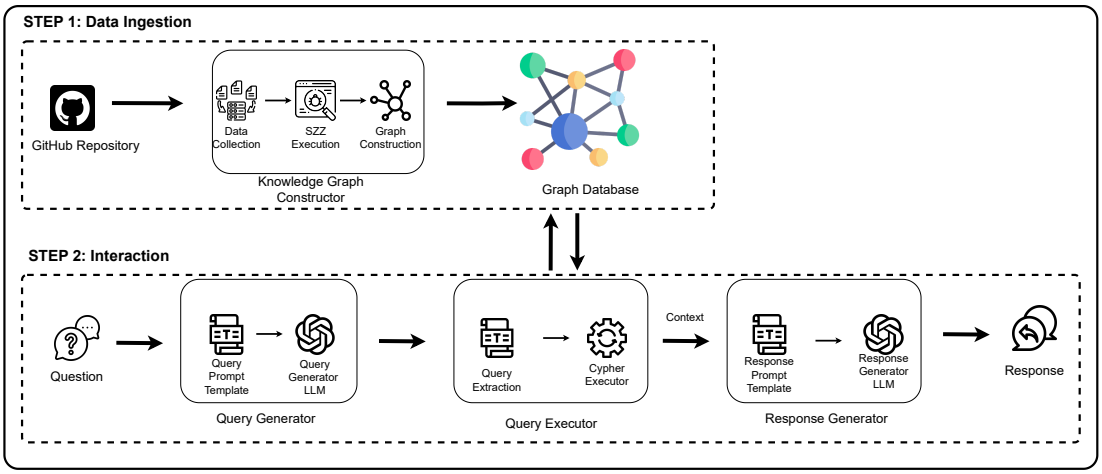

Fig. 1. Overview of the architecture of RepoChat [12]

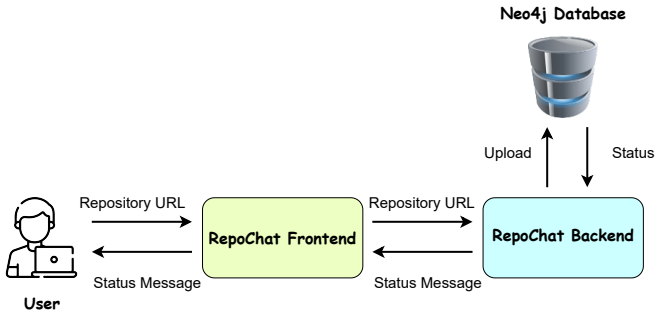

Fig. 2. Overview of Data Ingestion workflow

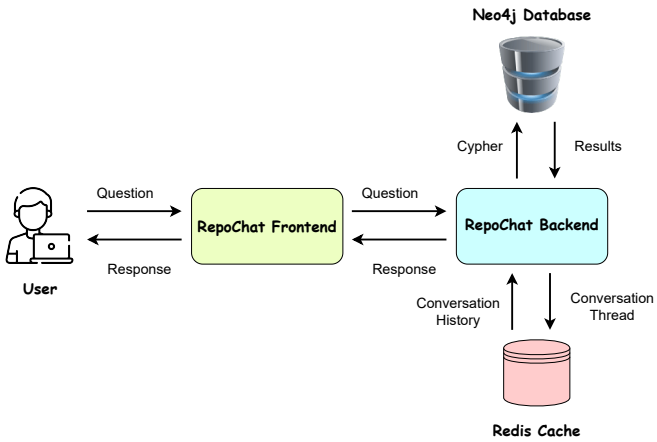

Fig. 3. Overview of Interaction workflow

The endpoint returns this user-friendly response, completing the interaction cycle.

B. Frontend

On the front end, RepoChat provides two user interfaces: the Data Ingestion UI and the interaction UI. The Data Ingestion UI allows users to input the necessary information for the ingestion process, such as repository details and authentication credentials. It communicates with the Data Ingestion endpoint to initiate the data collection and knowledge graph construction. The interaction UI offers a chat-like environment where users can input their questions and receive responses. It manages session variables and interacts with the chat endpoint, providing a seamless and intuitive user experience. Figure 4 shows the user interface of the interaction step with RepoChat.

C. Deployment

RepoChat is deployed as a web application accessible on https://repochattool.streamlit.app/ through standard web browsers. Both the backend and frontend components are hosted on a server. Also, it is available publicly on Zenodo [1] for users to clone and run locally on their projects.

IV. Evaluation

In this section, we evaluate the accuracy of RepoChat in answering software repository-related questions. Our evaluation focuses on the tool's ability to provide correct answers to user queries. We did not assess user experience in this study.

We evaluated RepoChat by conducting a user study to get real-world users to interact with the tool. We begin the evaluation by constructing a knowledge graph of a GitHub project to allow the users to interact with the tool. The approach implemented in RepoChat is generalizable to GitHub software projects that include issue IDs specified in the commit logs of their bug-fixing commits (e.g., *"fixes issue #123"*), which is necessary for the execution of the SZZ algorithm [18]. We randomly selected the "management-sdk-js" project for our evaluation. This project meets the criteria of having issue IDs specified in the commit logs of bug-fixing commits and is a real-world project. We first completed the Data Ingestion step of our approach, collecting the commit history, issue data, files, and user information for the "management-sdk-js" project. We

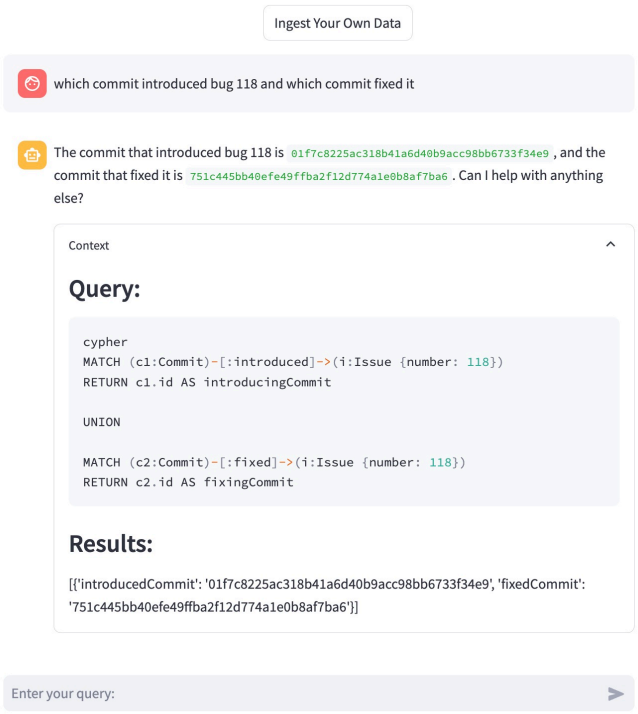

Fig. 4. UI showing a user's question, the response from the chatbot, and the context. The question was asked on the "management-sdk-js" project

then constructed the knowledge graph for the repository and stored it in a graph database.

We evaluated RepoChat's accuracy in responding to user questions by conducting a user study with four participants during the Interaction step of RepoChat. These participants are graduate students with a median of four years of experience working with GitHub repositories. Each participant was asked to complete ten tasks, which represent the intents in repository-related questions presented by Abdellatif et al. [13]. Each participant asked ten questions corresponding to the ten tasks. We logged the questions, the generated Cypher queries, the results from executing the Cypher queries, and the final responses generated to the questions. We compared the responses to the actual information in the repository to evaluate the accuracy of RepoChat.

RepoChat achieved an accuracy of 90%, correctly answering 36 out of the 40 questions. The four questions that RepoChat failed to answer were the Task 8 questions posed by all four participants, which sought to identify the developer with the highest number of unfixed bugs. For this question, we intended to find developers who have been assigned bugs that are still open. However, the language model in RepoChat interpreted the query as finding users who have created issues that are open, leading to incorrect answers.

Overall, the 90% accuracy achieved by RepoChat demonstrates its capability to accurately answer repository-related questions.

V. Related Works

Chatbots have been developed to automate various software engineering tasks. For instance, Xu et al. [20] and Cai et al. [21] both proposed a bot, AnswerBot, to provide answer summary from multiple StackOverflow posts to users' questions. Abdellatif et al. [6] introduced MSRbot, a chatbot layered on top of software repositories to answer common development and maintenance questions. Similarly, Farhour et al. [22] proposed AlphaBot, which automates the query annotation process using weak supervision, improving the natural language understanding (NLU) component of SE chatbots and enhancing their performance. Abedu et al. [8] evaluated an LLM-based chatbot for answering repository-related questions and found that the approach inaccurately retrieved data for response generation, leading to poor performance.

Recent research suggests that integrating knowledge graphs with LLMs can enhance their capabilities by providing structured, factual knowledge that LLMs often lack [23]. In the software engineering domain, knowledge graphs have been used to model software repositories, enabling complex queries and deeper insights [24], [25]. Abu-Rasheed et al. [26] demonstrated that using knowledge graphs as a source of factual context for LLM prompts reduces hallucinations and improves the precision of generated content. This work is an implementation of our approach in [12].

VI. Conclusion and Future Work

In this paper, we introduced **RepoChat**, a web-based tool designed to answer repository-related questions by synergizing large language models (LLMs) with knowledge graphs. RepoChat addresses the limitations of LLM-based chatbots for software repository-related questions by accurately interpreting user questions and retrieving relevant data from software repositories to answer them. RepoChat operates in two steps—the Data Ingestion step and the Interaction step—to effectively collect repository metadata, construct a comprehensive knowledge graph, and utilize LLMs to translate natural language questions into executable graph queries.

Our evaluation of RepoChat achieved a 90% accuracy in providing correct answers to repository-related questions. This high level of accuracy highlights the effectiveness of combining LLMs with knowledge graphs to enhance the responses in software repository question answering. The tool not only simplifies the process of accessing and extracting meaningful insights from repository data but also makes this information more accessible to non-technical stakeholders who may lack expertise in using APIs or complex query languages.

For future work, we plan to extend RepoChat's capabilities by integrating LLM agents into the tool. This enhancement will enable RepoChat to perform actions such as opening and closing issues, as well as assisting in the code review process of the project.

References

[1] S. Abedu, L. Menneron, S. Khatoonabadi, and E. Shihab, "RepoChat:an LLM-powered chatbot for GitHub repository question-answering [code]." [Online]. Available: https://zenodo.org/records/14673950

[2] M. Vidoni, "A systematic process for Mining Software Repositories: Results from a systematic literature review," *Information and Software Technology*, vol. 144, p. 106791, Apr. 2022.

[3] A. E. Hassan, "The road ahead for Mining Software Repositories," in *2008 Frontiers of Software Maintenance*. IEEE, Sep. 2008, pp. 48–57.

[4] A. Begel and T. Zimmermann, "Analyze this! 145 questions for data scientists in software engineering," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: Association for Computing Machinery, May 2014, pp. 12–23.

[5] V. S. Sharma, R. Mehra, and V. Kaulgud, "What Do Developers Want? An Advisor Approach for Developer Priorities," in *2017 IEEE/ACM 10th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE Press, May 2017, pp. 78–81.

[6] A. Abdellatif, K. Badran, and E. Shihab, "MSRBot: Using bots to answer questions from software repositories," *Empirical Software Engineering*, vol. 25, no. 3, pp. 1834–1863, May 2020.

[7] S. Banerjee and B. Cukic, "On the Cost of Mining Very Large Open Source Repositories," in *2015 IEEE/ACM 1st International Workshop on Big Data Software Engineering*. IEEE Press, May 2015, pp. 37–43.

[8] S. Abedu, A. Abdellatif, and E. Shihab, "LLM-Based Chatbots for Mining Software Repositories: Challenges and Opportunities," in *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '24. New York, NY, USA: Association for Computing Machinery, Jun. 2024, pp. 201–210.

[9] A. Abdellatif, D. Costa, K. Badran, R. Abdalkareem, and E. Shihab, "Challenges in Chatbot Development: A Study of Stack Overflow Posts," in *Proceedings of the 17th International Conference on Mining Software Repositories*, ser. MSR '20. New York, NY, USA: Association for Computing Machinery, Sep. 2020, pp. 174–185.

[10] E. Adamopoulou and L. Moussiades, "An Overview of Chatbot Technology," in *Artificial Intelligence Applications and Innovations*, I. Maglogiannis, L. Iliadis, and E. Pimenidis, Eds. Cham: Springer International Publishing, 2020, pp. 373–383.

[11] C. Shah, R. W. White, R. Andersen, G. Buscher, S. Counts, S. S. S. Das, A. Montazer, S. Manivannan, J. Neville, X. Ni, N. Rangan, T. Safavi, S. Suri, M. Wan, L. Wang, and L. Yang, "Using Large Language Models to Generate, Validate, and Apply User Intent Taxonomies," May 2024.

[12] S. Abedu, S. Khatoonabadi, and E. Shihab, "Synergizing LLMs and knowledge graphs: A novel approach to software repository-related question answering," Dec. 2024. [Online]. Available: http://arxiv.org/abs/2412.03815

[13] A. Abdellatif, K. Badran, D. E. Costa, and E. Shihab, "A Comparison of Natural Language Understanding Platforms for Chatbots in Software Engineering," *IEEE Transactions on Software Engineering*, vol. 48, no. 8, pp. 3087–3102, Aug. 2022.

[14] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–5, May 2005.

[15] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu, "A Survey on Knowledge Graphs: Representation, Acquisition, and Applications," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 2, pp. 494–514, Feb. 2022.

[16] R. Xie, Z. Liu, J. Jia, H. Luan, and M. Sun, "Representation Learning of Knowledge Graphs with Entity Descriptions," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, Mar. 2016.

[17] OpenAI, "Best practices for prompt engineering with the OpenAI API," https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-the-openai-api, 2024.

[18] D. A. da Costa, S. McIntosh, W. Shang, U. Kulesza, R. Coelho, and A. E. Hassan, "A Framework for Evaluating the Results of the SZZ Approach for Identifying Bug-Introducing Changes," *IEEE Transactions on Software Engineering*, vol. 43, no. 7, pp. 641–657, Jul. 2017.

[19] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo, X. Zhou, M. Chenhao, G. Li, K. Chang, F. Huang, R. Cheng, and Y. Li, "Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs," *Advances in Neural Information Processing Systems*, vol. 36, pp. 42 330–42 357, Dec. 2023.

[20] B. Xu, Z. Xing, X. Xia, and D. Lo, "AnswerBot: Automated generation of answer summary to developers' technical questions," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Oct. 2017, pp. 706–716.

[21] L. Cai, H. Wang, B. Xu, Q. Huang, X. Xia, D. Lo, and Z. Xing, "AnswerBot: An answer summary generation tool based on stack overflow," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, Aug. 2019, pp. 1134–1138.

[22] F. Farhour, A. Abdellatif, E. Mansour, and E. Shihab, "A Weak Supervision-Based Approach to Improve Chatbots for Code Repositories," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, pp. 105:2378–105:2401, Jul. 2024.

[23] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu, "Unifying Large Language Models and Knowledge Graphs: A Roadmap," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 7, pp. 3580–3599, Jul. 2024.

[24] Y. Zhao, H. Wang, L. Ma, Y. Liu, L. Li, and J. Grundy, "Knowledge Graphing Git Repositories: A Preliminary Study," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, Feb. 2019, pp. 599–603.

[25] A. Malik, B. Adams, and A. Hassan, "Towards graph-anonymization of software analytics data: Empirical study on JIT defect prediction," *Empirical Software Engineering*, vol. 29, no. 4, p. 76, Jun. 2024.

[26] H. Abu-Rasheed, C. Weber, and M. Fathi, "Knowledge Graphs as Context Sources for LLM-Based Explanations of Learning Recommendations," Mar. 2024.