IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, November 2026
Impact Factor: 8.2

Repoxray: A Retrieval-Augmented Framework for Semantic Analysis and Repository-Level Understanding of Software Codebases

Hrishita Gohain[1], Nidhi Pote[2], Manasi Rugle[3], Shehnaz Siddique[4]
Usha Mittal Institute of Technology, Santacruz, Mumbai, India[1-4]
[1] hrishita.gohain@gmail.com,[2] nidhi.umit@gmail.com
[3] manasirugle2414@gmail.com,[4] work.shehnazsiddique@gmail.com

Abstract: *Understanding an unfamiliar software repository is difficult because modern codebases contain interconnected files, modules, dependencies, and configuration artifacts. Traditional keyword-based search provides limited contextual understanding, while directly processing large repositories using Large Language Models (LLMs) introduces context and scalability challenges.*
This paper presents RepoXray, a retrieval-augmented framework for repository-level software understanding. The system combines file-level LLM analysis, semantic embeddings, vector retrieval, and Map-Reduce-based synthesis to generate repositorylevel architecture and security-oriented reports. It also provides an AI Chat Console that retrieves relevant repository information to answer natural-language queries, along with an interactive File Explorer for inspecting source code and AI-generated file-level analysis.
RepoXray was evaluated using 20 public GitHub repositories across four programming languages. It achieved 84.91% architecture fact coverage and 84.75% repository-aware question-answering accuracy. The preliminary combined score was 84.83%. Security performance could not be quantitatively validated because the evaluation dataset contained no positive ground-truth vulnerability instances. These results demonstrate the potential of RepoXray for improving repository understanding while highlighting the need for further evaluation.

Keywords: Software Repository Analysis, Retrieval-Augmented Generation, Large Language Models, Semantic Search, Code Embeddings, Repository Understanding.

I. INTRODUCTION

Software repositories have become increasingly complex as applications are developed using multiple modules, frameworks, libraries, configuration files, services, and external dependencies. Open-source repositories hosted on platforms such as GitHub may contain hundreds or thousands of files, making it difficult for developers to understand an unfamiliar project efficiently. Repository understanding is essential for software maintenance, debugging, onboarding, architecture comprehension, and modification of existing systems.

Traditional repository exploration relies heavily on directory browsing, documentation, and keyword-based search. Keyword search is useful when the user knows the exact identifier or textual phrase being searched for, but it does not directly represent semantic relationships between a natural-language question and the implementation that answers it. Code representation models such as CodeBERT demonstrated that programming language and natural language can be represented in a shared learned space for tasks such as natural-language code search [1]. GraphCodeBERT further demonstrated the value of incorporating structural information, such as data flow, into code representation learning [2]. Recent advances in Large Language Models (LLMs) have expanded the possibilities for code understanding and

Copyright to IJARSCT
www.ijarsct.co.in
DOI: 10.48175/IJARSCT-40821

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, November 2026
Impact Factor: 8.2

software engineering. However, repository-level tasks remain challenging because relevant information is distributed across multiple files and components. SWE-bench demonstrated that realistic GitHub software-engineering tasks frequently require coordination across functions, classes, and files rather than isolated code generation [3].

Retrieval-Augmented Generation (RAG) provides a mechanism for combining an LLM with external information retrieved at inference time. The original RAG formulation combines parametric language-model knowledge with a nonparametric dense vector index [4]. In software engineering, repository-level retrieval has become an active research area because selecting useful repository context is difficult when the available codebase is substantially larger than the model's effective context.

RepoXray approaches repository understanding by combining file-level LLM analysis, semantic embeddings, vector retrieval, and repository-level synthesis to generate architectural and security-oriented information and support natural-language exploration of the analyzed codebase.

The main contributions of this work are:

- A multi-stage framework for repository-level semantic analysis using LLM-based file processing and
- repository-level synthesis.
- Integration of semantic embeddings and vector retrieval with repository-aware natural-language querying.
- A Map–Reduce analysis process that combines file-level observations into repository-level architecture and
- security-oriented reports.
- Integration of an AI Chat Console and File Explorer for interactive repository exploration.
- An evaluation using architecture facts and repository-aware question-answering across multiple repositories.

II. RELATED WORK

A. Code Representation and Semantic Search

Code search traditionally relies on lexical matching between a query and source code. Neural code representation models introduced semantic representations that connect natural-language descriptions with programming-language artifacts. CodeBERT is a Transformer-based model trained using programming-language and natural-language data [1]. GraphCodeBERT extended this direction by incorporating data-flow information into code representation learning [2].

RepoXray uses the general principle of representing repository content as vectors. Instead of training a new code representation model, it uses an embedding service to represent repository content and perform semantic retrieval.

B. Retrieval-Augmented Generation

RAG was introduced as a method for combining a generative model with externally retrieved information [4]. Research surveys identify retrieval, augmentation, and generation as major components of RAG systems and highlight challenges such as retrieval quality, hallucination, and evaluation [5].

For repository-level software engineering, RAG provides a mechanism for selecting a subset of repository information instead of providing an entire codebase to an LLM.

C. Repository-Level Code Understanding

RepoCoder introduced an iterative retrieval-generation approach for repository-level code completion [6]. RepoBench subsequently provided a benchmark for repository-level code auto-completion [7].

SWE-benchevaluatesLLMsonrealsoftwareengineeringissuescollectedfromGitHubrepositories[3].

CodexGraphinvestigates graph-based repository interaction [8], while CodeRAG investigates query construction, multiple retrieval paths, and reranking [9].

The survey by Tao et al. identifies scalability, reliability, retrieval quality, and context construction as continuing challenges in repository-level retrieval-augmented systems [10].

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, November 2026
Impact Factor: 8.2

TABLE I
LITERATURE REVIEW OF RELATED APPROACHES

Paper Title	Methodology	Observations and Remarks
CodeBERT [1]	Transformer-based NL-code representation	Supports semantic code search and motivates semantic representation in RepoXray.
GraphCodeBERT [2]	Code representation using data-flow information	Shows the value of structural information for code understanding.
RepoCoder [6]	Iterative retrieval and generation	Demonstrates the importance of retrieving cross-file repository context.
RepoBench [7]	Repository-level retrieval and completion benchmark	Highlights the need for repository-level evaluation.
SWE-bench [3]	Real GitHub issues requiring repositorylevel reasoning	Shows the difficulty of reasoning across multiple files and components.
RAG Survey [5]	Survey of retrievalaugmented generation	Identifies retrieval quality and reliability challenges.
CodexGraph [8]	Graph-based repository representation	Shows the benefits of structural retrieval.
CodeRAG [9]	Query construction, multi-path retrieval, and reranking	Highlights the importance of retrieval and context selection.

III. PROBLEM STATEMENT AND RESEARCH OBJECTIVES

A. Problem Statement

A software repository contains heterogeneous information distributed across source files, configuration files, documentation, and modules. A developer investigating an unfamiliar repository must identify relevant files, understand their individual roles, infer relationships between components, and answer questions about the overall system.

Keyword-based retrieval can locate textual matches but may fail when the user's query and the relevant implementation use different terminology. Conversely, providing an entire repository to an LLM can exceed practical context limits or introduce large amounts of irrelevant information.

The research problem addressed by RepoXray is:

How can LLM-based semantic analysis and retrieval be combined to provide useful repository-level architectural understanding, security-oriented observations, and natural-language interaction with software codebases?

B. Research Objectives

The objectives of the system are:

1. To automatically acquire and preprocess GitHub repositories.
2. To generate structured file-level semantic and analytical information.
3. To synthesize file-level information into repository-level reports.
4. To store semantic representations for similarity-based retrieval.
5. To support repository-aware natural-language questions through a RAG-based interface.
6. To provide interactive file exploration with AI-generated analysis.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, November 2026
Impact Factor: 8.2

7. To evaluate the system using predefined architecture and question-answering test cases.

C. Research Questions

The study considers the following research questions:

- RQ1: How effectively does RepoXray recover predefined architectural facts from analyzed repositories?
- RQ2: How accurately does the repository-aware RAG component answer predefined natural-language
- questions?
- RQ3: What limitations arise from semantic retrieval and LLM-generated repository analysis?
- RQ4: What additional evaluation is required to establish scalability and generalizability?

IV. PROPOSED METHODOLOGY

RepoXray follows a multi-stage repository-analysis pipeline consisting of repository acquisition, preprocessing, file-level analysis, semantic embedding generation, vector storage, repository-level synthesis, and natural-language retrieval. The methodology is designed to analyze repository components individually and subsequently combine the generated information to support repositorylevel understanding and querying.

The overall repository-analysis and retrieval workflow is illustrated in Fig. 1.

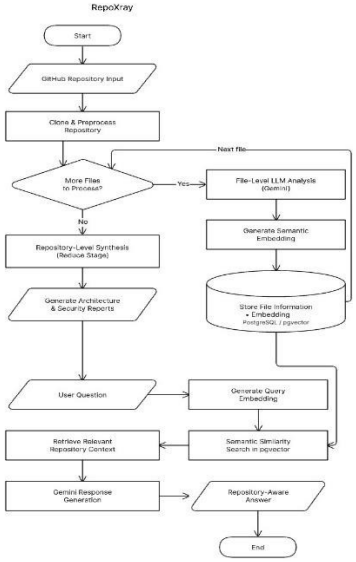


Fig. 1. RepoXray Repository Analysis and Retrieval Workflow

A. Repository Acquisition

The user provides a GitHub repository URL. The system clones the repository into a processing environment and extracts its file and directory structure. Repository acquisition forms the first stage of the analysis pipeline.

B. Repository Preprocessing

The repository is inspected to identify relevant source files and exclude files that do not contribute useful information to code understanding. This preprocessing stage reduces unnecessary processing and prevents irrelevant repository artifacts from dominating subsequent analysis.

C. File-Level Analysis

Repository processing is divided into independent file-level tasks. Each worker processes an individual source file using an LLM and produces structured information, including a file summary and security-oriented observations. This forms the Map stage of the analysis process.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, November 2026
Impact Factor: 8.2

The Map stage avoids treating the complete repository as a single large prompt. Instead, the repository is decomposed into smaller units that can be processed independently [3,6].

D. Semantic Embedding Generation

The processed repository content is converted into vector representations using the configured Gemini embedding service. Each embedding represents semantic properties of its associated repository content in a vector space.

These representations allow repository content to be compared based on semantic similarity rather than requiring exact lexical matches [1,2].

E. Vector Storage

RepoXray stores repository information and its vector representations in PostgreSQL using the pgvector extension. The stored embeddings provide the basis for similarity-based repository retrieval during natural-language interaction.

F. Reduce-Stage Repository Synthesis

After file-level processing is completed, a master synthesis stage combines the individual analysis results. The Reduce stage generates repository-level architecture information and a consolidated security-oriented report.

This Map–Reduce organization allows local file-level observations to be produced first and subsequently combined to obtain a broader representation of the repository.

G. Natural-Language Retrieval

The AI Chat Console provides a repository-aware natural-language query interface. A user submits a question, which is converted into a query embedding. The system performs similarity search against the stored repository embeddings and retrieves relevant repository information.

The retrieved information is combined with repository architecture context before being passed to the LLM for response generation.

The resulting workflow can be represented as:

$$q \rightarrow E(q) \rightarrow \text{SimilaritySearch} \rightarrow C \rightarrow LLM(C, q) \qquad (1)$$

where q represents the user query, $E(q)$ represents its embedding, C represents the retrieved repository context, and $LLM(C, q)$ represents the generated repository-aware answer.

H. Repository Reports

The system produces repository-level architecture information and security-oriented analysis. The architecture report summarizes the organization and responsibilities of repository components, while the security report aggregates potential findings and categorizes them by severity.

The security output is intended as AI-assisted analysis and should not be interpreted as a replacement for a professional security audit.

V. SYSTEM ARCHITECTURE

The RepoXray architecture consists of a frontend, backend orchestration layer, repository-processing components, AI analysis services, persistent storage, and semantic retrieval components.

The frontend provides the repository-analysis interface, progress information, generated reports, file exploration, and the AI Chat Console. The backend coordinates repository acquisition, preprocessing, AI-based analysis, storage, and repositoryaware interaction through REST APIs.

The processing pipeline consists of repository ingestion, filtering, file-level LLM analysis, embedding generation, vector storage, repository-level synthesis, and semantic retrieval.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, November 2026
Impact Factor: 8.2

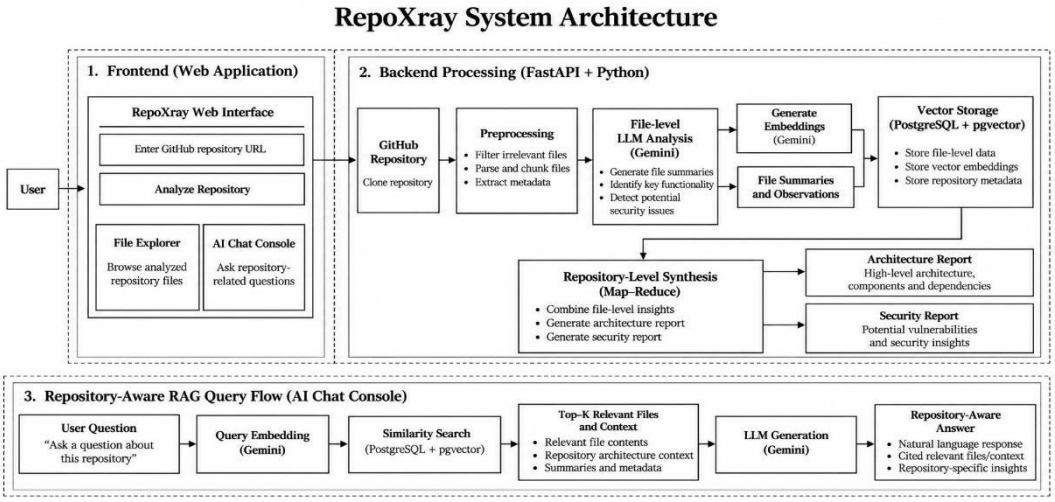


Fig. 2. Architecture of the RepoXray System

The conceptual architecture consists of the following stages:

1. User and frontend interface.
2. GitHub repository ingestion.
3. Repository filtering and preprocessing.
4. Parallel file-level analysis.
5. Gemini-based analysis and embedding generation.
6. PostgreSQL and pgvector storage.
7. Reduce-stage repository synthesis.
8. Architecture and security-oriented reports.
9. Query embedding and semantic retrieval.
10. Gemini-based repository-aware response generation.

VI. IMPLEMENTATION

A. Frontend

The frontend is implemented using Next.js, React, and TypeScript. It provides the user interface for submitting repositories, monitoring analysis progress, viewing generated reports, exploring analyzed files, and interacting with the AI Chat Console.

B. Backend

The backend is implemented using FastAPI and Python. It coordinates repository acquisition, preprocessing, file-level analysis, repository-level synthesis, storage, and retrieval. It also provides REST interfaces used by the frontend.

C. Authentication

Firebase Authentication is used for user authentication. Server-side verification is used to authenticate requests and maintain separation of repository data between users.

D. Repository Processing

GitHub repositories are cloned and processed file by file. Repository content that does not contribute useful information to code understanding is filtered before AI analysis.

Processing repositories at the file level also supports analysis of information distributed across multiple files [3,6].

E. LLM Analysis

Google Gemini is used to analyze repository files. File-level processing generates structured summaries and potential securityoriented observations.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, November 2026
Impact Factor: 8.2

After individual files have been processed, a synthesis stage combines these outputs to generate repository-level architecture and security-oriented reports.

F. Embeddings and Vector Retrieval

Semantic embeddings are generated for repository content and stored in PostgreSQL using pgvector. During natural-language querying, the user's query is converted into an embedding and compared with stored repository representations to retrieve relevant information.

G. File Explorer

The File Explorer provides searchable access to analyzed repository files. Users can select a file to inspect its original source code together with its AI-generated analysis. This connects repository-level understanding with fine-grained inspection of individual implementation files.

H. AI Chat Console

The AI Chat Console implements a repository-aware RAG workflow. It allows users to ask natural-language questions about the analyzed repository.

The RepoXray query workflow is represented as:

User Question → Query Embedding → Vector Search → Relevant Files → Repository Context → Gemini → Answer

For each natural-language question, the system generates a query embedding and performs similarity search against the stored repository representations. Relevant repository information is retrieved and supplied as context for response generation.

I. Evaluation Design

The evaluation of RepoXray was conducted using 20 public GitHub repositories across four programming languages: Python, Java, JavaScript, and TypeScript.

The evaluation focused on two principal metrics:

1. Architecture Fact Coverage.
2. Repository-Aware RAG Question-Answering Accuracy.

A separate security evaluation was also considered. However, the evaluation dataset did not contain positive ground-truth vulnerability instances. Therefore, security performance was reported as Not Applicable (N/A).

J. Architecture Fact Evaluation

Architecture understanding was evaluated by comparing the facts generated by RepoXray against predefined architecture facts for each repository.

Across the 20 evaluated repositories, a total of 464 architecture facts were considered. RepoXray correctly covered 394 facts.

The architecture fact coverage is calculated as:

$$\text{Architecture Coverage} = \times 100 = 84.91\% \quad (2)$$

K. RAG Question-Answering Evaluation

The repository-aware question-answering capability was evaluated using 177 questions distributed across the 20 repositories.

Out of 177 questions, 150 were answered correctly.

The RAG question-answering accuracy is calculated as:

$$\text{RAG Accuracy} = \times 100 = 84.75\% \quad (3)$$

L. Security Evaluation

The security evaluation is reported as Not Applicable (N/A) and is excluded from the overall evaluation score.

The evaluation dataset did not contain positive ground-truth vulnerability instances. Therefore, precision, recall, and F1-score for positive vulnerability detection could not be meaningfully established.

Consequently, the security component is considered an AI-assisted analysis mechanism requiring further validation using datasets containing confirmed vulnerabilities.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, November 2026
Impact Factor: 8.2

VII. RESULTS

A. Evaluation Results

The evaluation results of RepoXray are summarized in Table II.

TABLE II
REPOXRAY EVALUATION RESULTS

Metric	Result
Repositories Evaluated	20
Programming Languages	4
Architecture Facts Evaluated	464
Architecture Facts Covered	394
Architecture Fact Coverage	84.91%
RAG Questions Evaluated	177
RAG Questions Passed	150
RAG QA Accuracy	84.75%
Security Evaluation	N/A
Overall Evaluation Score	84.83%

The overall evaluation score is calculated as the mean of Architecture Fact Coverage and RAG QA Accuracy. Security evaluation is excluded from this calculation.

B. Architecture Understanding

RepoXray achieved an architecture fact coverage of 84.91%, correctly identifying 394 out of 464 predefined architecture facts across 20 repositories.

This result indicates that the Map-Reduce analysis approach can recover useful architectural information from different software repositories.

C. Repository-Aware Question Answering

The RAG-based question-answering module achieved an accuracy of 84.75%, with 150 correct responses out of 177 evaluated questions.

This demonstrates the ability of RepoXray to retrieve relevant repository context and generate answers to natural-language questions about the codebase.

However, the remaining incorrect responses indicate scope for improvement in context retrieval and answer generation.

D. Preliminary Combined Score

The overall evaluation score is calculated as the arithmetic mean of Architecture Fact Coverage and RAG QuestionAnswering Accuracy.

$$S = \frac{84.91 + 84.75}{2} = 84.83\% \qquad (4)$$

The resulting score represents the preliminary combined performance of the two evaluated metrics. It should not be interpreted as the overall accuracy of the complete system because the two metrics measure different aspects of repository understanding.

The security evaluation is excluded because the available dataset did not contain positive ground-truth vulnerability instances.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, November 2026
Impact Factor: 8.2

E. Semantic Retrieval

Semantic retrieval plays an important role in connecting natural-language queries with relevant repository information. It allows users to ask questions about concepts such as authentication, database configuration, architecture, and dependencies without knowing the exact file names or identifiers.

By converting queries and source code representations into vector embeddings, RepoXray uses similarity-based retrieval to identify relevant repository context.

However, retrieval quality remains an important factor in generating accurate responses. Selecting incomplete or less relevant context can affect the quality of the final answer.

F. Architecture and Security Analysis

The Map-Reduce architecture provides an effective approach to separating file-level analysis from repository-level synthesis. Individual source files are processed independently, and their results are combined to generate a global architecture overview.

The security analysis component generates potential security observations using Large Language Models. However, its detection performance could not be quantitatively validated because the evaluation dataset did not contain positive ground-truth vulnerability instances.

Therefore, the security output should be considered an AI-assisted analysis mechanism rather than a fully validated security scanner.

VIII. CONCLUSION AND FUTURE WORK

This paper presented RepoXray, a retrieval-augmented framework designed to improve software repository understanding through AI-based analysis and semantic retrieval. The system combines file-level LLM analysis, semantic embeddings, vector storage, and Map-Reduce-based synthesis to generate repository-level architecture and security-oriented insights.

The integration of the AI Chat Console further extends the system by allowing developers to interact with analyzed repositories through natural-language questions. The File Explorer provides access to individual source files along with their AIgenerated analysis, making repository exploration more interactive and accessible.

The evaluation conducted across 20 public repositories covering four programming languages achieved 84.91% architecture fact coverage and 84.75% RAG question- answering accuracy. The preliminary combined score was 84.83%. However, security detection performance could not be quantitatively established due to the absence of positive ground-truth vulnerability instances.

Overall, RepoXray demonstrates the potential of combining LLM-based analysis and retrieval techniques to support repository level software understanding. Further evaluation and improvements in retrieval quality, security validation, and scalability can strengthen the system in future work.

ACKNOWLEDGMENT

The authors thank the Department of Computer Science and Technology, Usha Mittal Institute of Technology, for its support throughout this work.

REFERENCES

[1] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, “Codebert: A pre- trained model for programming and natural languages,” in Findings of the Association for Computational Linguistics: EMNLP 2020, 2020.

[2] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, M. Tufano, S. K. Deng, C. Clement, D. Drain, N. Sundaresan, J. Yin, D. Jiang, and M. Zhou, “Graphcodebert: Pre-training code representations with data flow,” in International Conference on Learning Representations, 2021.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, November 2026
Impact Factor: 8.2

[3] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. R. Narasimhan, "Swe-bench: Can language models resolve real-world github issues?" in The Twelfth International Conference on Learning Representations, 2024.
[4] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Ku¨ttler, M. Lewis, W.-t. Yih, T. Rocktaschel, S. Riedel, and¨ D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in Advances in Neural Information Processing Systems, vol. 33, 2020.
[5] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," arXiv preprint arXiv:2312.10997, 2023.
[6] F. Zhang, B. Chen, Y. Zhang, J. Keung, J. Liu, D. Zan, Y. Mao, J.-G. Lou, and W. Chen, "Repocoder: Repository-level code completion through iterative retrieval and generation," in Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, 2023.
[7] T. Liu, C. Xu, and J. McAuley, "Repobench: Benchmarking repository-level code auto-completion systems," in The Twelfth International Conference on Learning Representations, 2024.
[8] X. Liu, B. Lan, Z. Hu, Y. Liu, Z. Zhang, F. Wang, M. Q. Shieh, and W. Zhou, "Codexgraph: Bridging large language models and code repositories via code graph databases," in Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), 2025.
[9] S. Zhang, Y. Ding, S. Lian, S. Song, and H. Li, "Coderag: Finding relevant and necessary knowledge for retrieval-augmented repository-level code completion," in Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, 2025.
[10] Y. Tao, Y. Qin, and Y. Liu, "Retrieval-augmented code generation: A survey with focus on repository-level approaches," arXiv preprint arXiv:2510.04905, 2026.