WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

Codegraph AI: AI-Based Codebase Analysis System

Ms V Gayathiri[#1], Mrs M Sabeetha[#2], Dhanya S[#3], Dhakshitha S[#3], Dinesh Sriram Velan T[#4], Darsha Nivaash S A[#5], Anu Shri J [#7]

[#1] Guide, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India
[#2] Co-Ordinator, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India
[#3] Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India
[#4] Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and TechnologyIndia.
[#5] Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology,India .
[#6] Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology,India.
[#7] Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.

ABSTRACT

Software development involves analyzing large and complex codebases to understand program structure, identify errors, detect code duplication, and maintain software quality. Traditional code analysis techniques can require considerable manual effort and may not fully explain complex program behaviour. This paper presents an AI-Based Codebase Analysis System intended to support source-code analysis through automated techniques and AI-assisted code understanding. The system is designed to process source-code files, examine code structure, identify potential issues, summarize program components, and provide recommendations for improving readability and maintainability. Depending on the implemented tools and models, additional capabilities may include code-duplication checks and identification of potential security weaknesses. The proposed modular workflow consists of codebase input, preprocessing, code analysis, issue identification, and report generation. The system aims to reduce repetitive review effort and help developers understand unfamiliar projects more efficiently. The accuracy and usefulness of its outputs depend on the programming languages supported, the analysis tools and models used, and evaluation against representative codebases.

Keywords—Artificial Intelligence, Codebase Analysis, Static Analysis, Code Quality, Defect Detection, Code Summarization.

1. INTRODUCTION

Software development has become increasingly complex because applications are larger, use multiple programming languages, and must remain reliable and maintainable. A codebase contains source files, libraries, configuration files, and other resources that together form a software application. Understanding a large codebase can be challenging, especially when developers need to identify errors, recognize duplicated code, understand program functionality, and maintain existing systems.

Traditional codebase analysis commonly relies on manual code reviews, rule-based checks, and static analysis tools. These approaches are useful for identifying known patterns and coding-standard violations, but they may require substantial effort and can provide limited explanations of complex program structures. Research on machine learning for source code describes how learned techniques can support tasks involving code patterns and program understanding [1].

AI-based methods have expanded research into code representation and automated code understanding. For example, code2vec explores learned representations of code snippets [2], while CodeBERT studies pre-trained representations for programming and natural languages [3]. Research on large language models trained on code also investigates their capabilities and limitations on programming tasks [4]. These studies provide a foundation for exploring AI-assisted code analysis, but the output of any model should be checked rather than treated as automatically correct.

The AI-Based Codebase Analysis System aims to combine conventional code-analysis techniques with suitable AI methods to help developers inspect source code, understand program components, identify potential coding issues, and receive improvement suggestions. Depending on the actual implementation, the system may also support code-duplication checks, security-related checks, and explanations of relationships between modules. The goal is to organize these activities in a single workflow and present the findings in a clear report.

1.1 Background of Codebase Analysis

Codebase analysis is the process of examining source code to understand its structure, functionality, quality, and dependencies. A project may contain many files, functions, classes, libraries, and modules that work together. As projects grow, understanding how these components interact and maintaining them consistently becomes more difficult. Automated analysis can help organize this information and highlight areas that require developer attention [1].

1.2 Need for Artificial Intelligence in Software Development

AI techniques can learn patterns from code and may assist with code classification, summarization, and contextual explanations. Learned code representations and pre-trained models have been investigated for improving code-related tasks [2], [3]. However, AI-generated findings can be incomplete or incorrect; therefore, they should be combined with appropriate validation and conventional analysis methods.

1.3 Problem Statement

Modern software projects often contain large volumes of code distributed across multiple files and modules. Developers may need considerable time to understand unfamiliar applications and identify possible errors or maintainability concerns. Conventional tools can detect specific rule violations, but they may not provide all the explanations needed to understand code functionality. Therefore, a system is needed to organize source-code inspection, potential issue identification, code summaries, and recommendations in a consistent workflow.

1.4 Objectives of the Proposed System

1. Automated code analysis: examine source files and identify relevant structural information.
2. Potential bug identification: highlight suspicious code patterns using the implemented analysis methods.
3. Code summarization: provide understandable summaries of functions, classes, or other components where supported.
4. Code-quality improvement: suggest changes that may improve readability, organization, and maintainability.
5. Code-duplication detection: identify repeated or similar code segments if this feature is implemented.
6. Security-related analysis: flag potential weaknesses when supported by the selected tools. Report generation: present findings and recommendations in a clear, structured format.

1.5 Scope of the System

An AI-Based Codebase Analysis System introduces an intelligent framework for maintaining architectural integrity across massive enterprise software ecosystems. It bridges the critical operational gap between complex, multi-language code repositories and automated compliance validation engines. By continually contextualizing codebase intent, semantic dependencies, and underlying system vulnerabilities, the platform provides engineering teams with proactive security safeguards. This proactive posture ensures that fast-moving development teams can ship code rapidly without introducing architectural regressions or compliance risksIt transforms passive version-control systems into predictive engines that secure, optimize, and document software assets.

2. LITERATURE SURVEY

2.1 Review of Existing Research

Research into machine learning for source code examines how learning-based methods can support code-related tasks and the study of code naturalness [1]. This body of work provides broad context for applying AI to software development, while also showing that task definitions and evaluation methods matter.

Alon et al. introduced code2vec, which learns distributed representations of code from paths in abstract syntax trees and uses those representations for code-related prediction tasks [2]. Feng et al. presented CodeBERT, a pre-trained model designed to represent programming and natural languages [3]. Chen et al. evaluated large language models trained on code, examining their performance on programming tasks [4]. Together, these studies illustrate several approaches to learning from source code.

2.2 Existing Approaches and Techniques

Static analysis examines source code without executing the program and can identify rule violations or certain potential defects. Rule-based approaches apply predefined patterns and checks. Machine-learning approaches learn associations from code examples, while deep-learning and pre-trained-model approaches learn richer representations that can support code-related tasks [1]–[4]. These methods differ in their input requirements, supported tasks, explainability, and dependence on training data or model configuration.

2.3 Review of Key Findings

- Automated defect detection: learning-based methods can help identify code patterns associated with potential defects, but findings require validation.
- Code representation: learned representations can capture useful information from source code and support downstream tasks [2], [3].
- AI-assisted developer support: AI methods may assist with code understanding and summarization, but their outputs can vary by task and input [4].
- Complementary techniques: conventional static checks and AI-based methods can serve different roles in a code-analysis workflow.

2.4 Research Gap

Existing research covers code representation, model-based code understanding, and programming-task evaluation [1]–[4]. A practical system may still need to organize input handling, preprocessing, code checks, understandable findings, and report generation into one workflow. The proposed system focuses on this integration. The precise research gap should be validated against a broader review of relevant studies and against the features that are actually implemented and evaluated.

3. METHODOLOGY

This section describes the overall process for the AI-Based Codebase Analysis System. The workflow is organized into existing analysis techniques and the proposed integrated method. The exact analysis capabilities depend on the tools, models, and programming languages supported by the implementation.

3.1 Existing Methodology

Existing code-analysis workflows commonly use manual review, static analysis, and rule-based inspection. Manual review relies on a developer reading and interpreting the code. Static analysis examines source code without execution to identify issues covered by its rules. Machine-learning and deep-learning methods can be used for selected code-related tasks, depending on the available model and data [1]–[4]. These approaches provide useful capabilities but may operate as separate tools and may require developers to interpret results from multiple sources.

3.2 Proposed Methodology

The proposed workflow combines codebase input, preprocessing, code analysis, issue identification, and report generation. AI-based methods are used where they are available and appropriate; conventional analysis methods can provide complementary checks.

1. **Codebase Input:** The user submits supported source-code files or selects a project directory for analysis.
2. **Code Preprocessing:** The system validates the input and prepares supported files for analysis. Where supported, it extracts components such as functions, classes, and structural information.
3. **Code Analysis:** Static-analysis rules and suitable AI-based methods examine the source code for relevant patterns, structural information, or potential quality concerns.
4. **Issue Identification:** The system organizes possible bugs, duplication, maintainability concerns, or security-related findings only where the implemented tools support those checks.
5. **Summary and Recommendation Generation:** The system prepares code summaries and improvement suggestions when these functions are supported by the selected models or tools.
6. **Report Generation:** The system presents findings, explanations, and recommendations in a structured report for developer review.
7. **Developer Verification:** The developer checks the findings and tests any proposed changes before applying them.

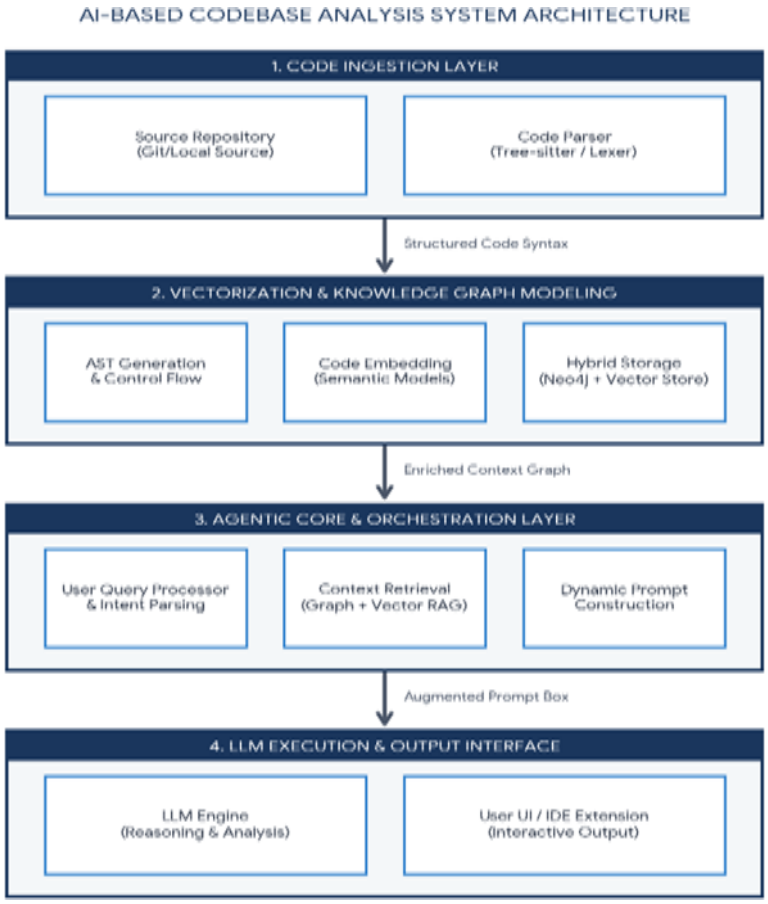
3..3 System Architecture

The system architecture consists of an input module, preprocessing module, analysis engine, result-organizing module, and reporting interface. The input module receives source files. The preprocessing module validates and prepares them. The analysis engine applies the available static-analysis tools and AI models. The result-organizing module groups findings and summaries, and the reporting interface presents them to the user. A practical implementation should also handle unsupported files and analysis errors.

Workflow: Source-Code Input → Preprocessing and Validation → Static / AI-Based Analysis → Issue and Summary Generation → Report Presentation → Developer Verification.

3.4 Workflow of the Proposed System:fig(a)

1. Collect supported source-code files or a project directory.
2. Validate files and prepare code for analysis.
3. Analyze code with available static-analysis methods and suitable AI techniques.
4. Organize detected patterns, potential issues, and supported summaries.
5. Generate a structured report with findings and suggestions.
6. Allow the developer to review findings and verify changes through testing.



Fig(a)

3.5 Advantages of the Proposed Methodology

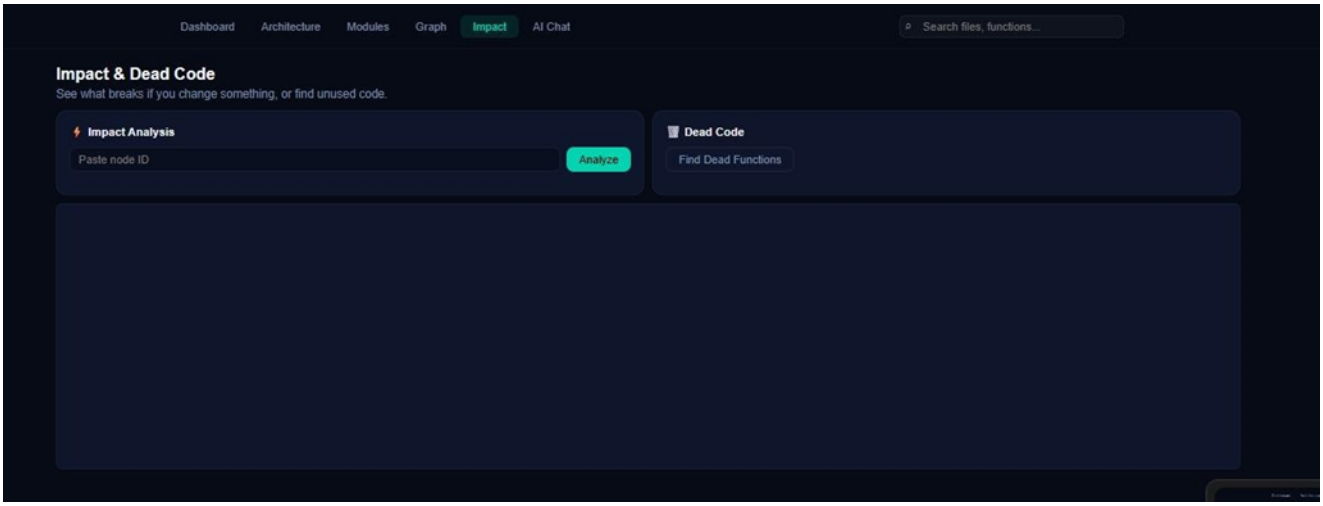
- Can reduce repetitive manual inspection for supported analysis tasks.
- Can help developers navigate unfamiliar codebases.
- May support earlier identification of potential defects.
- Can present code observations and recommendations in a structured report.

4. RESULTS AND DISCUSSION

The proposed AI-Based Codebase Analysis System is intended to process source code and present organized information about the analyzed project. Depending on the features implemented, the output may include code summaries, potential issue reports, code-quality observations, duplication findings, or recommendations for improvement.

4.1 System Outputs

The expected report structure includes the input files analyzed, supported analysis results, descriptions of identified concerns, and recommendations where available. Findings should distinguish confirmed rule violations from AI-generated suggestions or potential issues. Unsupported features should not be presented as completed capabilities:fig(b)



Fig(b)

4.2 Evaluation and Discussion

The usefulness of the system should be evaluated using representative codebases and clearly defined criteria. Suitable measures may include the number of correctly identified issues, false-positive and false-negative findings, processing time, and developer assessment of the clarity of generated summaries. The supported programming languages, test cases, tools, and model versions should be reported so that results can be interpreted and reproduced:fig(c)

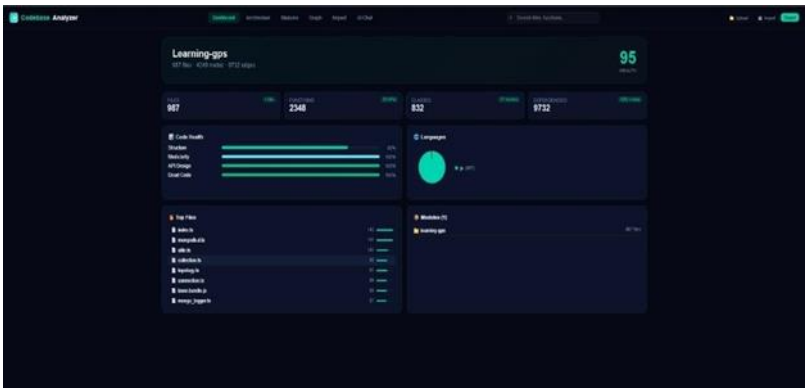


Fig (c)

5. CONCLUSION

This paper presents an AI-Based Codebase Analysis System designed to support the understanding, review, and maintenance of software codebases. The proposed workflow organizes code input, preprocessing, analysis, issue identification, summary generation, and reporting. By combining conventional analysis techniques with suitable AI-based methods, the system aims to help developers review code more efficiently and understand unfamiliar projects. The reliability of its output depends on the implemented tools, supported languages, and validation process. Future work may include broader language support, improved explanations, integration with development environments, and systematic evaluation on representative projects. Any future improvements should be assessed through experiments before claims about accuracy or efficiency are made.

REFERENCES

[1] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, “A Survey of Machine Learning for Big Code and Naturalness,” ACM Computing Surveys, vol. 51, no. 4, Article 81, 2018, doi: 10.1145/3212695.

[2] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, “code2vec: Learning Distributed Representations of Code,” Proceedings of the ACM on Programming Languages, vol. 3, POPL, Article 40, 2019, doi: 10.1145/3290353.

[3] Z. Feng et al., "CodeBERT: A Pre-Trained Model for Programming and Natural Languages," in Findings of the Association for Computational Linguistics: EMNLP 2020, pp. 1536–1547, 2020, doi: 10.18653/v1/2020.findings-emnlp.139.

[4] M. Chen et al., "Evaluating Large Language Models Trained on Code," arXiv preprint arXiv:2107.03374, 2021. Available: https://arxiv.org/abs/2107.03374.