A Synopsis of Project on

C^3 (Code.Collab.Commit): A Collaborative Code Editor using Repository Level LLM.

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Engineering

in

Computer Science and Engineering Data Science

by

Rohan Waghode(21107008)
Meet Jamsutkar(22207004)
Arya Patil(21107009)
Urvi Padelkar(211070054)

Under the Guidance of

Prof. Anagha Aher

Department of Computer Science and Engineering (Data Science)
A.P. Shah Institute of Technology
G.B.Road,Kasarvadavli, Thane(W)-400615
UNIVERSITY OF MUMBAI

Academic Year 2024-2025

Approval Sheet

This Project Synopsis Report entitled ***"C³ (Code.Collab.Commit): A collaborative Code Editor using Repository Level LLM."*** submitted by ***Rohan Waghode (21107008), Meet Jamsutkar (22207004), Arya Patil (21107009), Urvi Padelkar (21107054)***
is approved for the partial fulfillment of the requirement for the award of the degree of ***Bachelor of Engineering*** in ***Computer Science and Engineering - Data Science*** from the ***University of Mumbai***.

Prof. Anagha Aher
Guide

Prof. Anagha Aher
HOD, Computer Science and Engineering-Data Science

Place:A.P.Shah Institute of Technology, Thane
Date:

CERTIFICATE

This is to certify that the project entitled **"C^3 (Code.Collab.Commit): A collaborative Code Editor using Repository Level LLM"** submitted by **"Rohan Waghode" (21107008)**, **"Meet Jamsutkar" (22207004)**, **"Arya Patil" (21107009)**, and **"Urvi Padelkar" (21107054)** for the partial fulfillment of the requirement for the award of a degree Bachelor of Engineering in Information Technology to the University of Mumbai, is a bonafide work carried out during the academic year 2024-2025.

Prof. Anagha Aher
Project Guide

Prof. Anagha Aher
HOD, CSE-Data Science

Dr. Uttam D.Kolekar
Principal

External Examiner(s)

1.

2.

Internal Examiner(s)

1.

2.

Place:A.P.Shah Institute of Technology, Thane
Date:

Acknowledgement

We have great pleasure in presenting the synopsis report on **"C3 (Code.Collab.Commit): A collaborative Code Editor using Repository Level LLM" .** We take this opportunity to express our sincere thanks towards our guide **Prof. Anagha Aher** for providing the technical guidelines and suggestions regarding line of work. We would like to express our gratitude towards his constant encouragement, support and guidance through the development of project.

We thank **Prof. Anagha Aher** Head of Department for his encouragement during the progress meeting and for providing guidelines to write this report.

We express our gratitude towards BE project co-ordinator **Prof. Anagha Aher**, for being encouraging throughout the course and for their guidance.

We also thank the entire staff of APSIT for their invaluable help rendered during the course of this work. We wish to express our deep gratitude towards all our colleagues of APSIT for their encouragement.

Rohan Waghode
(21107008)

Meet Jamsutkar
(22207004)

Arya Patil
(21107009)

Urvi Padelkar
(21107054)

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, We have adequately cited and referenced the original sources. We also declare that We have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Rohan Waghode (21107008)

Meet Jamsutkar (22207004)

Arya Patil (21107009)

Urvi Padelkar (21107054)

Date:

Abstract

The collaborative code editor aims to enhance individual and team productivity by addressing the challenges of distributed development teams. It facilitates seamless, real-time collaboration, enabling developers to work together effortlessly from any location. With repository-level intelligent code completion powered by Large Language Models (LLM), it minimizes coding errors and streamlines workflows, making managing large projects easier. Additionally, the editor features automated code analysis, which helps developers navigate complex codebases efficiently, reducing the time spent on manual reviews. This not only boosts workflow efficiency but also allows teams to focus on critical tasks for faster development cycles. By offering smart, contextual assistance tailored to specific code and repositories, the editor improves accuracy and simplifies workflows, making it ideal for remote teams. Designed to meet the demands of complex software projects, it creates an integrated environment that enhances collaboration, reduces inefficiencies, and promotes high-quality code development. The editor also includes features like semantic search, smart commits, and automated documentation, ensuring optimal code management. Built using Electron JS and FastAPI, the platform is designed for scalability, offering real-time synchronization and intelligent features that make it ideal for multi-user editing, version control, and remote collaboration in distributed teams.

Keywords: *Collaborative Code Editor, Large Language Model, Repository Level Code Generation, Smart Commits, Intelligent Code Completion.*

Contents

List of Figures

List of Tables

List of Abbreviations

LLM:	Large Language Model
AI:	Artificial Intelligence
OT:	Operational Transform
CRDT:	Conflict-Free Replicated Data Type
AWS:	Amazon Web Services
GCP:	Google Cloud Platform
HDFS:	Hadoop Distributed File System
DRF:	Django Rest Framework
ML:	Machine Learning
API:	Application Programming Interface

Chapter 1

Introduction

As the complexity of software development projects continues to grow, developers are facing increasing challenges in maintaining productivity and efficiency, especially in distributed team environments. Modern development teams, often spread across various geographical locations, encounter difficulties in collaborating seamlessly and ensuring the accuracy of their work. Traditional code editors and version control tools, while functional for basic tasks, are inadequate for supporting real-time, synchronous collaboration and intelligent, context-aware code suggestions. These shortcomings hinder the overall workflow and can lead to significant inefficiencies, including duplicated efforts, missed updates, and unresolved code conflicts.

One of the critical issues is the lack of robust tools that allow multiple developers to work on the same codebase simultaneously without experiencing delays or conflicts. Existing tools rely on asynchronous communication, which can result in out-of-date code versions or errors that aren't identified until later in the process. This is particularly problematic for large-scale projects with frequent updates, where seamless collaboration is essential to maintaining project momentum.

Current code editors do not offer the level of intelligent assistance needed to manage the complexity of today's software systems. While autocomplete features exist, they are often limited to basic syntax and common patterns, without understanding the broader context of the project. Developers are frequently forced to navigate large codebases manually, searching for references or documentation, which is both time-consuming and error-prone. Without intelligent code suggestions that consider repository-level context, developers are left vulnerable to making mistakes that could have been easily avoided with better tooling.

Besides to collaboration challenges, the time-consuming nature of code reviews further compounds the problem. Manual reviews of large, complex codebases require significant effort from senior developers and team leads, which delays the overall development cycle. Intelligent code analysis tools are needed to automate aspects of this process, identifying potential issues or areas for improvement before human reviewers even see the code. By streamlining the review process, teams can significantly reduce development times while improving code quality.

Another area that lacks attention in current tools is the management of codebases. Large projects with many contributors require advanced systems that can handle complex operations like semantic search, automated documentation, and smart commits. These features help developers navigate the intricacies of large-scale projects, ensuring that the right information is accessible when needed. Current systems, however, often fall short, providing only

basic search and commit functionalities that do not take into account the full scope of the project.

The need for a more integrated and intelligent solution has never been more pressing. A collaborative code editor that offers real-time, multi-user editing, intelligent repository-level code completion, and automated code analysis would revolutionize the development process. By addressing the limitations of current tools, such a system would allow developers to focus on innovation and problem-solving rather than spending valuable time resolving conflicts, searching through code, or conducting manual reviews.

1.1 Motivation

The primary motivation for this project arises from the growing difficulties encountered by developers in distributed teams. With the rise of remote and hybrid work models, software development teams are often dispersed across various locations and time zones. This trend creates an urgent need for development tools that can support real-time, multi-user collaboration without suffering from typical issues such as delays, miscommunication, or synchronization conflicts. According to Zhang et al. [11], existing collaboration tools, while functional to some extent, fail to meet the specific needs of modern development environments, particularly in real-time collaboration and intelligent code management.

The limitations of current tools include:

- **Inefficient collaboration tools:** Distributed teams often struggle with real-time collaboration, as many existing platforms are unable to provide smooth, context-aware interactions. These tools frequently lack the capability to reflect changes immediately or support seamless multi-user editing, resulting in communication breakdowns and delays in project delivery.
- **Lack of intelligent code management:** Current development environments often miss intelligent features like context-aware code suggestions or real-time repository analysis. This deficiency leads to higher error rates and longer review cycles. Without such capabilities, developers spend more time manually searching for code snippets or debugging, thereby reducing overall productivity and code quality.
- **Need for an integrated environment:** The increasing complexity of software projects necessitates a system that integrates collaboration and intelligent code management. An integrated platform would streamline workflows and reduce errors by ensuring seamless collaboration while automatically managing code analysis, documentation, and version control.

In light of these challenges, this project seeks to build upon the foundations established by Zhang et al. [11] in their work on repository-level code completion. The "RepoCoder" system utilizes iterative retrieval and generation techniques to enhance code completion across repositories. Our approach aims to iterate and improve upon this by integrating real-time collaboration features, intelligent context-aware suggestions, and automated documentation generation, thus creating a more cohesive development environment.

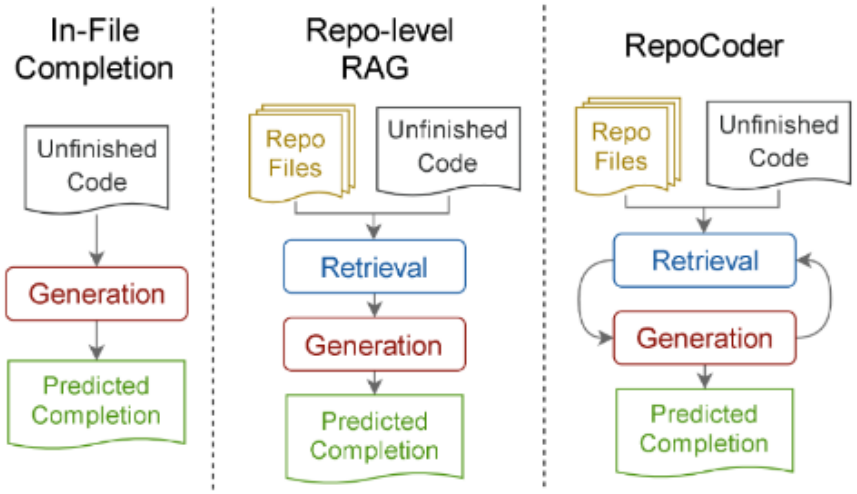

Figure 1.1: RepoCoder Architecture:Building upon the iterative retrieval and generation framework proposed by Fengji Zhang et al. 2023 [11]

By addressing these pain points and leveraging insights from RepoCoder, this project aims to enhance both individual and team productivity, ultimately contributing to more efficient software development practices.

1.2 Problem Statement

The central problem addressed by this project is the inefficiency and lack of intelligent collaboration tools in distributed software development environments. As software projects become increasingly complex and teams grow more geographically dispersed, traditional development tools are no longer adequate for maintaining productivity and ensuring code quality. Developers often face challenges in areas such as real-time collaboration, code management, and error detection, which can result in project delays, miscommunication, and coding inefficiencies.

- **Real-time Collaboration Challenges:** Current development tools rely heavily on asynchronous communication, making real-time collaboration between developers difficult. Distributed teams, spread across various locations and time zones, face issues related to simultaneous editing of the same codebase. Without a robust system for handling real-time changes and conflict resolution, code merges become time-consuming and error-prone. These synchronization issues often result in duplicated efforts, outdated code versions, and unresolved merge conflicts, which significantly hinder team productivity. Existing tools that attempt to support real-time collaboration either lack the necessary scalability for large teams or suffer from high latency, which disrupts workflow and creates inefficiencies in the development process.

- **Lack of Advanced Code Completion and Management Tools:** While traditional code editors provide basic code autocompletion features, they are often limited in scope, offering suggestions that fail to consider the full context of the project. Developers working on large-scale, complex projects frequently find themselves manually

searching through extensive codebases to locate relevant code snippets or documentation. This not only increases the risk of errors but also wastes valuable development time. The absence of repository-level code suggestions, semantic search, and intelligent autocompletion tailored to the project's specific needs forces developers to rely on repetitive tasks and inefficient workflows. Moreover, the need for more sophisticated codebase management tools is critical to maintaining consistency across large projects with multiple contributors, ensuring that code quality and adherence to coding standards are maintained.

- **Inefficient Code Review Processes:** Manual code reviews are another significant bottleneck in the development cycle, especially in larger projects. Senior developers or team leads must spend substantial time reviewing code manually, looking for errors, stylistic issues, or opportunities for optimization. This manual process is not only time-consuming but also susceptible to human error. Without automated tools for intelligent code analysis and error detection, many issues remain undetected until later in the development process, leading to costly delays and rework. Automating parts of the code review process would allow teams to detect and resolve issues earlier, improving both code quality and development speed.

- **Limited Codebase Management and Documentation:** Managing large, complex codebases across distributed teams is a challenging task, particularly when multiple contributors are working on different parts of the same project. Features such as semantic search, automated documentation, and smart commits are often lacking in traditional development tools. Developers struggle to find relevant information within a codebase, and documenting code manually is an often-neglected task that results in poor maintenance and onboarding experiences. As projects grow, it becomes increasingly important to implement systems that automate codebase management, ensuring that relevant code is easily accessible, documentation is always up to date, and commits are well-organized.

- **Scalability and Integration Issues:** As teams and projects scale, existing tools may not handle the increased load efficiently. Many platforms fail to support a large number of concurrent users or projects, leading to slow performance, data loss, or workflow disruptions. Additionally, integrating multiple tools and technologies for collaboration, code management, and version control can be challenging, especially when the tools are not designed to work seamlessly together. Developers are often forced to juggle multiple platforms, leading to context switching and reduced focus, which further affects productivity.

1.3 Objectives

In recent years, the increasing complexity of software projects and the rise of distributed development teams have highlighted the need for more advanced tools that enhance collaboration, streamline workflows, and provide intelligent support throughout the coding process. Traditional code editors and version control systems are no longer sufficient to meet the demands of modern software development, especially for large-scale projects involving multiple contributors. Therefore, the objectives of this project focus on addressing these challenges by developing a platform that integrates real-time collaboration, intelligent code generation,

and automated management features. This will ensure that developers can work efficiently, avoid common pitfalls, and maintain high-quality codebases, even in a distributed environment. The key objectives are:

- **Collaborative Platform for Real-time Simultaneous Code Editing:** To develop this it involves building a platform that allows multiple developers to simultaneously edit the same codebase in real-time. By leveraging socket-based streaming frameworks like Apache Kafka, changes made by any user are immediately propagated to all other participants, ensuring smooth collaboration with minimal latency and no conflicts.
- **Repository-Level Intelligent Code Generation and Completion:** To develop this objective it focuses on integrating advanced machine learning models, such as retrieval-augmented models, to provide intelligent code generation and autocompletion. These models can suggest relevant code snippets and auto-generate code based on the context, helping developers reduce coding errors and improve workflow efficiency.
- **Codebase Management with Semantic Search, Automated Documentation, and Smart Commits:** To implement a semantic search engine, developers can quickly locate relevant parts of the codebase. Automated documentation tools will generate and maintain up-to-date documentation, while smart commits will use AI to suggest commit messages based on changes made, improving codebase organization and version control practices.
- **Dashboard for Visualizing Individual and Team Performance Metrics:** To implement it involves creating a dashboard that tracks key metrics such as lines of code written, code quality, and bug detection for individual developers and teams. By visualizing these metrics, the dashboard provides insights into productivity and helps developers and managers optimize their coding practices and workflow.

1.4 Scope

This project focuses on developing a web-based collaborative code editor equipped with real-time editing capabilities and version control. A key feature of the platform is the integration of an LLM-based code completion system, designed to assist developers in writing efficient and high-quality code. Additionally, the system will include automated tools for code analysis and documentation, alongside a performance metrics dashboard. By supporting multiple programming languages and development environments, the platform aims to cater to a diverse user base, offering a comprehensive solution for modern software development workflows.

- **Web-based collaborative code editor:** The platform will be designed using Electron for building cross-platform desktop applications and Monaco Editor for providing a powerful, feature-rich code editor. It will cater to both real-time collaboration and individual code editing, allowing users to edit the same project concurrently while seeing live updates from others. This will be ideal for software development teams, coding bootcamps, and educational settings where collaboration and learning are key.

- **LLM-based code completion system:** By integrating advanced Large Language Models (LLMs), the system will offer context-aware code completion that adapts to the developer's coding style and project needs. It will support iterative code generation, providing real-time suggestions, fixing syntax errors, and even generating whole functions based on brief prompts or comments, thus boosting productivity and reducing human error.

- **Automated code analysis and documentation:** The system will include features for automated code analysis, documentation generation, and smart commits, ensuring code quality and easing the review process for development teams.

- **Scalable backend architecture:** The platform will be built on a robust, cloud-based infrastructure (AWS, Azure, or GCP) to handle a high volume of concurrent users and projects. Using FastAPI, the backend will ensure low-latency interactions, real-time synchronization, and scalability, enabling the platform to grow with the needs of teams ranging from small startups to large enterprises with complex requirements.

Chapter 2

Literature Review

The Literature Review explores various methodologies and frameworks that address challenges in collaborative programming, intelligent code generation, and real-time multi-user environments. The reviewed works include techniques like shared-locking for semantic conflict prevention, syntactic neural models for code generation, and repository-level code completion. These studies highlight key limitations such as complexity in conflict management, computational resource demands, and inefficiencies in documentation and retrieval systems. By analyzing these drawbacks, this review identifies gaps in existing solutions and provides insights into designing a more efficient, scalable, and intelligent collaborative code editor system.

2.1 Comparative Analysis of Recent study

The Collaborative Code Editor System Diagram presents a comprehensive framework designed to enhance collaborative coding practices within software development teams. By enabling real-time collaboration through synchronous editing, multiple developers can contribute to the same codebase simultaneously, with sophisticated conflict management mechanisms in place to address any discrepancies that may arise. The system leverages large language model (LLM)-based intelligent code completion and context-aware snippets, which serve to augment developer efficiency and minimize coding errors. Additionally, the platform emphasizes automated code analysis, providing real-time diagnostics that assist developers in identifying and resolving issues promptly. With features such as smart commit suggestions and robust version control—including Git integration—the system facilitates streamlined management of code changes across multiple users while ensuring a clear version history. Overall, this collaborative code editor serves as a valuable tool for modern software development teams, fostering effective communication and coordination while promoting productivity and maintaining high code quality.

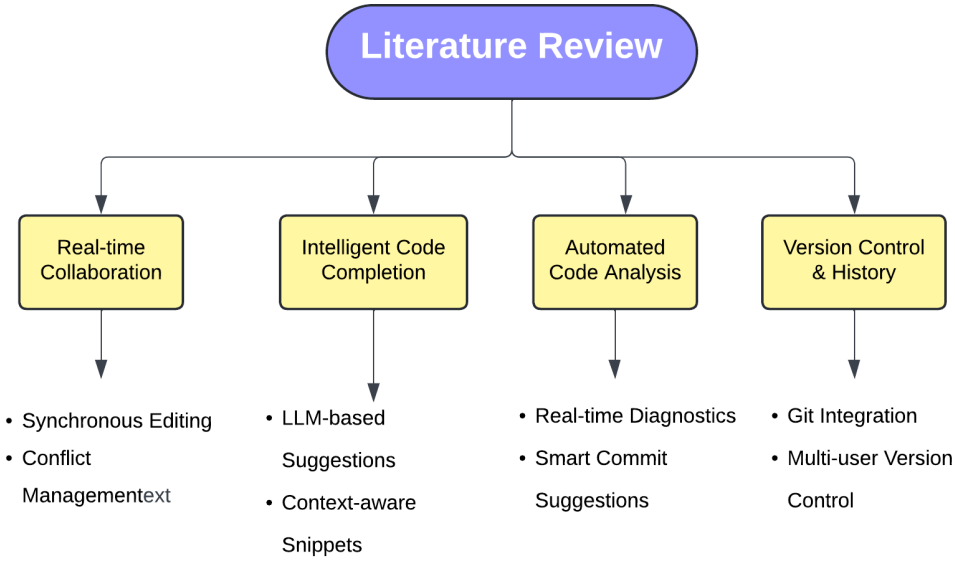

Figure 2.1: Categorization of literature survey

The development of collaborative code editors has focused on enabling real-time multi-user coding and knowledge sharing. These systems use operational transformation algorithms for resolving conflicts during simultaneous editing, as discussed in detail by researchers [8], allowing for seamless collaboration. A system aimed at enhancing code completion at the repository level was introduced, using iterative retrieval and generation techniques. This approach leverages the context of the entire repository to improve the accuracy of code suggestions. The authors [11] show how their method addresses high computational demands, outperforming traditional code completion systems.

A tool that automates README file generation by synthesizing relevant project information from the codebase has been developed. This method, which combines large language models with heuristic techniques, is highlighted by its ability to reduce manual documentation effort. The LARCH tool, as demonstrated in the study [6], was found to create coherent and useful README content with minimal input from developers. Automated documentation generation tools have been thoroughly evaluated for their efficiency, accuracy, and user satisfaction. The comparative analysis conducted by the authors [1] identifies the key features that make certain tools more effective in generating technical documentation, providing insights that could guide future improvements.

For collaborative multimedia design, a specialized revision control system for image editing has been introduced. This system allows multiple users to edit images simultaneously while maintaining version control, helping to prevent conflicts. The study [2] demonstrates how this system improves workflow efficiency in collaborative design projects.

In real-time collaborative programming, conflict prevention is crucial. A shared-locking mechanism has been proposed, allowing users to temporarily lock code segments to avoid semantic conflicts during simultaneous edits. The effectiveness of this approach is discussed by the authors [4], who show that it significantly enhances the collaboration experience for developers

An innovative approach to code generation uses natural language processing to retrieve relevant documentation and generate contextually appropriate code snippets. This method bridges the gap between documentation and coding, improving efficiency and accuracy, especially in complex coding scenarios, as shown in the work [12].

In the domain of semantic parsing, a framework that integrates generation and retrieval techniques has been introduced to improve retrieval accuracy. The research [10] demonstrates how this framework outperforms traditional retrieval methods, particularly in handling more complex parsing tasks.
In the paper [7] explore the integration of retrieval-augmented mechanisms within language models to enhance performance on various tasks. The authors discuss how their approach allows models to leverage external knowledge dynamically during inference, improving accuracy and adaptability in generating responses. They provide empirical evidence showing significant performance gains across multiple benchmarks.

According to the authors, well-maintained documentation is essential for enhancing software maintainability and usability, as highlighted in [6]. Their research underscores the importance of clear and comprehensive documentation, which helps ensure that software systems are easier to maintain, understand, and use. By prioritizing documentation efforts, developers can significantly improve the overall quality of their software.

To further explore the impact of documentation on software development, the table 2.1 provides a literature review summarizing various key findings on this topic.

Table 2.1: Comparative Analysis of Literature Survey

Sr. No	Title	Author(s)	Year	Methodology	Drawback
1	A revision control system for image editing in collaborative multimedia design	Fabio Calefato, Giovanna Castellano, and Veronica Rossano	2018	The process involves setting up automated tools like Mintlify, Document! X, Doxygen, Imagix, and JDocEditor for generating Java documentation.	Quality issues arise when documentation tools lack sophistication, potentially leading to inaccurate interpretations of complex code.
2	Shared-locking for semantic conflict prevention in real-time collaborative programming	Hongfei Fan et al.	2017	Shared-locking mechanism prevents semantic conflicts in real-time collaborative programming by locking specific sections of the code.	Excessive locking can reduce collaboration efficiency, leading to bottlenecks and delays.
3	Towards Automatic Generation of Short Summaries of Commits	S. Jiang, C. McMillan	2017	System for Automatic Generation of Short Summaries of Commits in software version control systems.	Struggles with large-scale codebases and complex changes, leading to less effective summarization.[5]

Sr. No	Title	Author(s)	Year	Methodology	Drawback
4	A Syntactic Neural Model for General-Purpose Code Generation	Pengcheng Yin, G. Neubig	2017	Syntactic Neural Model generates code from natural language descriptions by incorporating syntactic rules.	Dependency on predefined syntactic rules, limiting flexibility for less structured inputs.[9]
5	Generate-and-Retrieve: Use Your Predictions to Improve Retrieval for Semantic Parsing	Yury Zemlyanskiy et al.	2022	Generate-and-Retrieve method for improving retrieval in semantic parsing, combining a feedback loop with partially generated outputs.	Added complexity due to feedback loops and heavy dependence on the quality of initial predictions.
6	RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation	Fengji Zhang et al.	2023	Iterative retrieval and generation of code based on repository-level context.	High complexity and computational resource demands.
7	Context-based Operation Merging in Real-Time Collaborative Programming Environments	Hongguang Zhou et al.	2022	Merges concurrent operations based on contextual relevance in collaborative programming environments.	Errors in contextual interpretation can lead to incorrect merging, causing confusion or inconsistencies.
8	Context-Aware Code Completion Using Deep Learning Techniques	Alex L. Wang, Marina Alexe, Yoav Weiss	2021	Uses deep learning models for context-aware code completion in collaborative settings.	Training deep models is computationally expensive and requires large datasets.
9	Adaptive Multi-Agent Systems for Distributed Code Generation	J. M. Huang	2019	Adaptive multi-agent systems that work in distributed code generation environments.	Complexity in managing multiple agents and ensuring consistency in output.
10	Automatic Code Refactoring Using Natural Language Processing	A. Patel	2020	NLP-based system for automatic code refactoring.	Limited effectiveness in refactoring complex codebases with deep dependencies.
11	Automated Program Synthesis Using Reinforcement Learning [1]	K. Sato, P. Van der Laan	2019	Program synthesis via reinforcement learning models for high-level code generation.	Requires large training data and suffers from slow convergence during learning.
12	Real-Time Code Generation Using Probabilistic Models	L. X. Nguyen	2022	Probabilistic models for real-time collaborative code generation.	High computational overhead and limited scalability.
13	CoVSCode: a novel real-time collaborative programming environment for lightweight IDE	Hongfei Fan, Kun Li, Xiangzhen Li, Tianyou Song, Wenzhe Zhang, Yang Shi, Bowen Du	2019	CoVSCode provides a real-time collaborative programming environment that integrates lightweight IDE features with cloud-based collaboration tools to enhance user experience.	Limited support for complex project structures may affect usability in larger development environments. [3]

Chapter 3

Project Design

3.1 Proposed System Architecture

As the complexity of software development projects continues to grow, developers are facing increasing challenges in maintaining productivity and efficiency, especially in distributed team environments. Modern development teams, often spread across various geographical locations, encounter difficulties in collaborating seamlessly and ensuring the accuracy of their work. Traditional code editors and version control tools, while functional for basic tasks, are inadequate for supporting real-time, synchronous collaboration and intelligent, context-aware code suggestions. These shortcomings hinder the overall workflow and can lead to significant inefficiencies, including duplicated efforts, missed updates, and unresolved code conflicts.

The proposed system architecture is divided into four major components: (1) Collaborative Editor System Architecture, (2) Repository Level Code Generation, (3) Document Generation, and (4) Smart Commits. Each of these components addresses different critical aspects of the development process.

3.1.1 Collaborative Editor System Architecture

The first component focuses on real-time collaboration between developers. One of the critical issues is the lack of robust tools that allow multiple developers to work on the same codebase simultaneously without experiencing delays or conflicts. This is where the proposed collaborative system architecture as shown in fig. 3.1 comes into play, as depicted below.

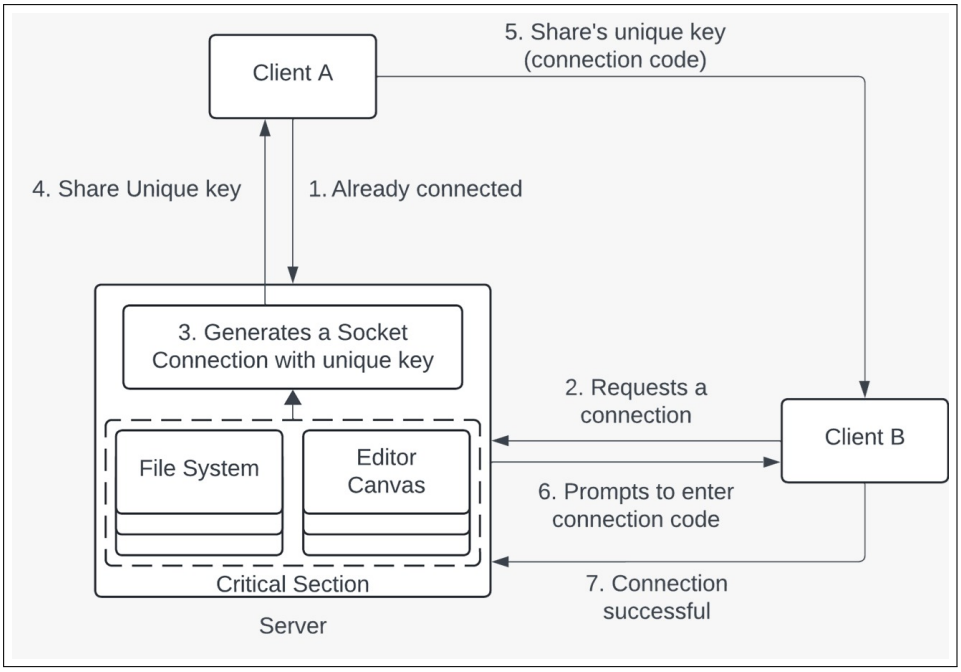

Figure 3.1: Collaborative Editor System Architecture

In the collaborative editor system, Client A connects to the server and generates a unique connection key. The server, which includes a file system and an editor canvas, establishes a socket connection for Client A. Once the connection key is generated, Client A shares it with Client B. Client B requests a connection by entering the unique connection key, after which the server authenticates and establishes a real-time connection for Client B. This setup allows both clients to collaborate synchronously in real time, working on the same codebase and using the file system and editor canvas.

The server manages file synchronization, edits, and conflict resolution to ensure seamless collaboration. The architecture is designed to address the inefficiencies of traditional asynchronous communication tools by allowing multiple users to interact with the same repository in real time.

3.1.2 Repository Level Code Generation

The second component addresses the automation of code generation at the repository level. In traditional development environments, developers often spend considerable time setting up repositories, generating repetitive code structures, and ensuring that new projects adhere to predefined architectural patterns. The proposed system fig. 3.2 streamlines this process by automating key aspects of repository setup and code generation.

Upon creating a new repository, the system provides developers with pre-defined templates based on the chosen language and framework. These templates include standard folder structures, configuration files, and skeleton code to accelerate the development process. Additionally, the system can generate boilerplate code for frequently used patterns, such as MVC (Model-View-Controller) structures in web development or service classes in backend systems.

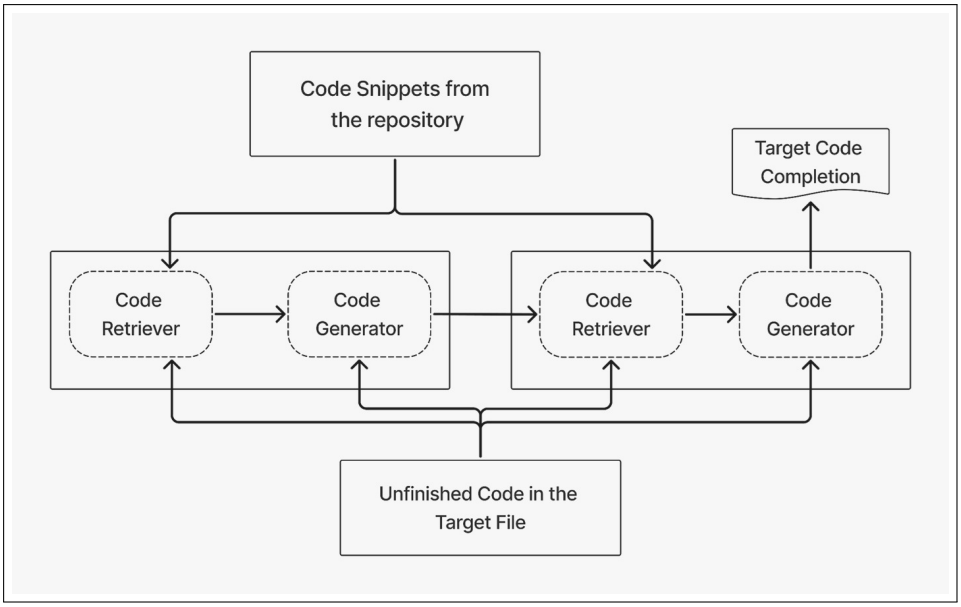

Figure 3.2: Repository Level Code Generation Architecture

The repository-level code generation also ensures that each repository adheres to organizational coding standards and practices. This feature reduces human error, speeds up the initial project setup, and ensures consistency across multiple projects.

3.1.3 Document Generation

The third component of the system architecture as shown in fig. 3.3 involves automated documentation generation. Developers often neglect documentation during the development process, leading to poor maintenance and difficult onboarding for new team members. The proposed system solves this issue by integrating real-time documentation generation based on the existing codebase.

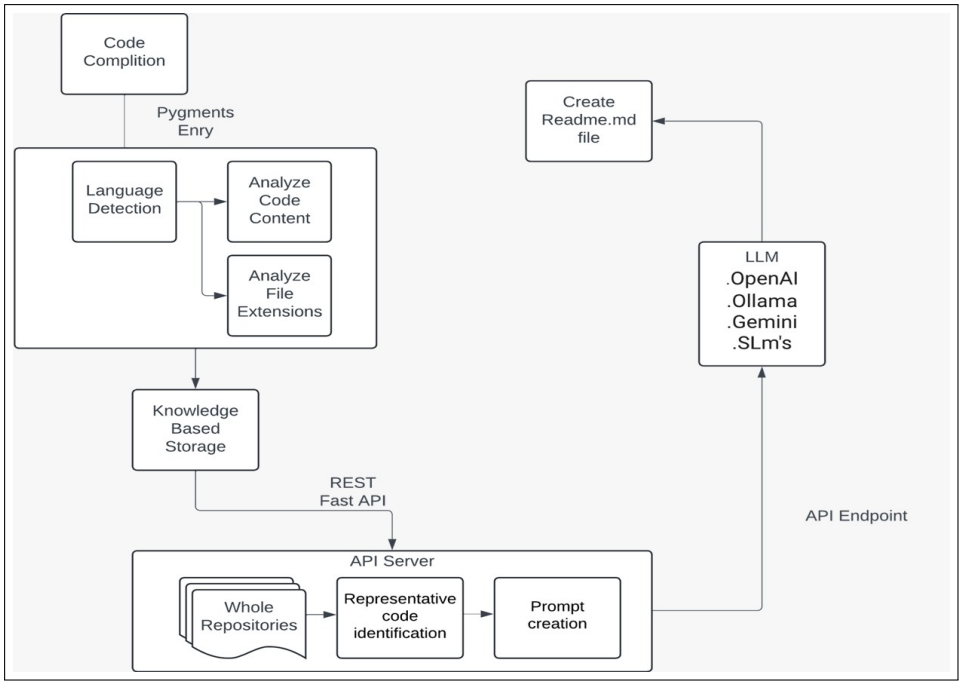

Figure 3.3: Document Generation System Architecture

Using natural language processing and intelligent parsing algorithms, the system scans the codebase and extracts relevant information to create detailed documentation for functions, classes, and modules. This documentation includes usage instructions, API details, and code dependencies, all formatted in a developer-friendly style.

Furthermore, the system supports version-controlled documentation, meaning that every time the code changes, the corresponding documentation is updated automatically. This ensures that the documentation remains in sync with the codebase, helping teams to maintain a reliable knowledge base throughout the project lifecycle.

3.1.4 Smart Commits

The fourth component focuses on the integration of "Smart Commits," an intelligent feature that automates the process of creating meaningful and contextual commit messages. Smart commits are generated automatically based on the code changes made in the repository, ensuring that every commit is accompanied by an appropriate and detailed message.

The system operates using an auto-commit scheduler, which monitors user activity and schedules commits if there is active participation every five minutes. This eliminates the need for manual commits and ensures that no changes are missed during the development process. The architecture of the smart commit feature is illustrated in Figure 3.4.

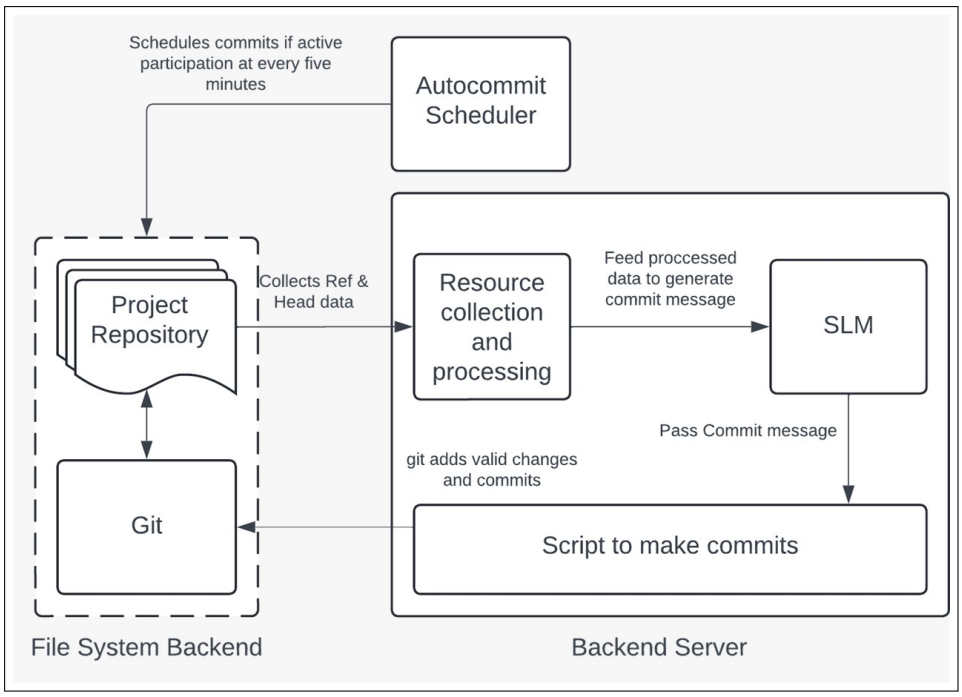

Figure 3.4: Smart Commit System Architecture

The smart commit architecture comprises the following key components:

- **Autocommit Scheduler**: Monitors user participation and schedules automatic commits.
- **Resource Collection and Processing**: Gathers reference and head data from the project repository to analyze changes and prepare the commit message.
- **SLM (Structured Language Model)**: Generates meaningful commit messages based on the processed data.
- **Script to Make Commits**: Executes the final commit to the repository, ensuring that the commit message and associated changes are recorded in Git.

This smart commit process ensures that all changes are saved in a structured and coherent manner, reducing the risk of inconsistent commit messages and helping to maintain a clean version history in the repository.

3.2 Data Flow Diagrams (DFD)

The Data Flow Diagram (DFD) in fig. 3.5 illustrates the flow of data within the system. It showcases how the user interacts with the system, particularly focusing on repository access, file management, code generation, and documentation handling. The DFD aids in visualizing the data exchange between various modules, helping to identify key areas such as collaboration, version control, and debugging.

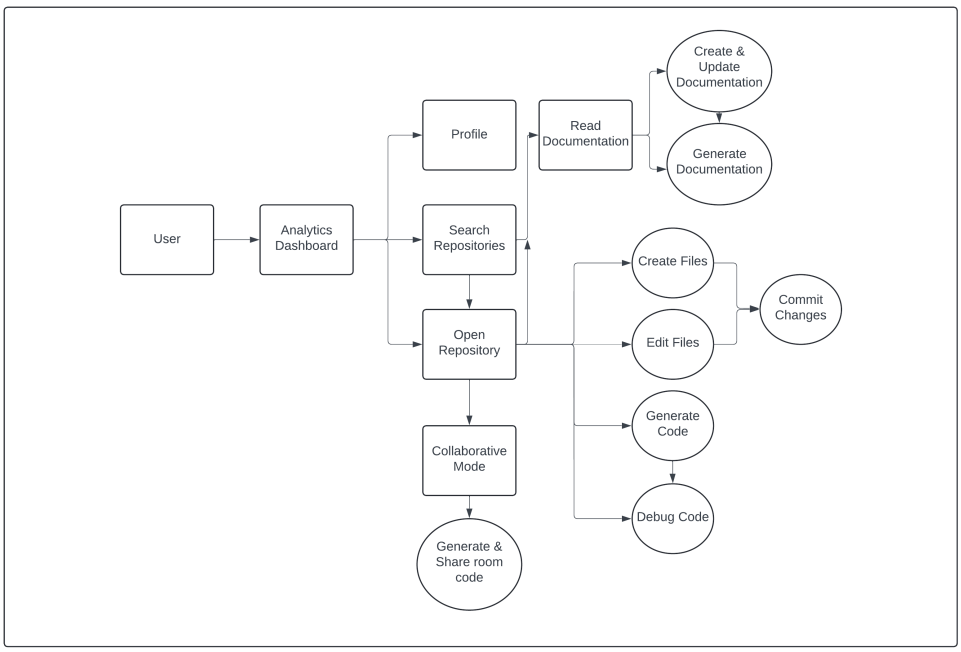

Figure 3.5: Data Flow Diagram of the Proposed System

In the proposed system, users can create and manage repositories, edit and commit changes, and generate documentation. The system will also handle file versioning and maintain a history of changes made by different users. Real-time collaboration is facilitated by synchronizing code updates across different instances of the editor, ensuring that users can work together without data conflicts.

Key data flow processes in the system include:

- **User Authentication**: Ensures secure access to the system.
- **Repository Management**: Handles repository creation, deletion, and permissions.
- **File Editing and Management**: Allows users to view, edit, and save files within the repository.
- **Code Compilation and Debugging**: Provides tools for compiling and debugging code within the editor.
- **Documentation Generation**: Automates the generation of documentation based on the codebase.
- **Commit and Version Control**: Tracks changes to the repository and facilitates versioning.

3.3 Use Case Diagrams

Use case diagrams are integral to system design as they depict the various interactions between the user and the system. By showing all possible scenarios that the user may encounter, these diagrams help in identifying and categorizing system requirements.

The use case diagram for this project, as shown in Figure 3.6, illustrates the int eraction between the user and the system. Key functionalities of the system include creating repositories, managing files, editing code, and generating documentation. The diagram also highlights the collaborative nature of the system, showing how multiple users can interact with the same repository simultaneously.

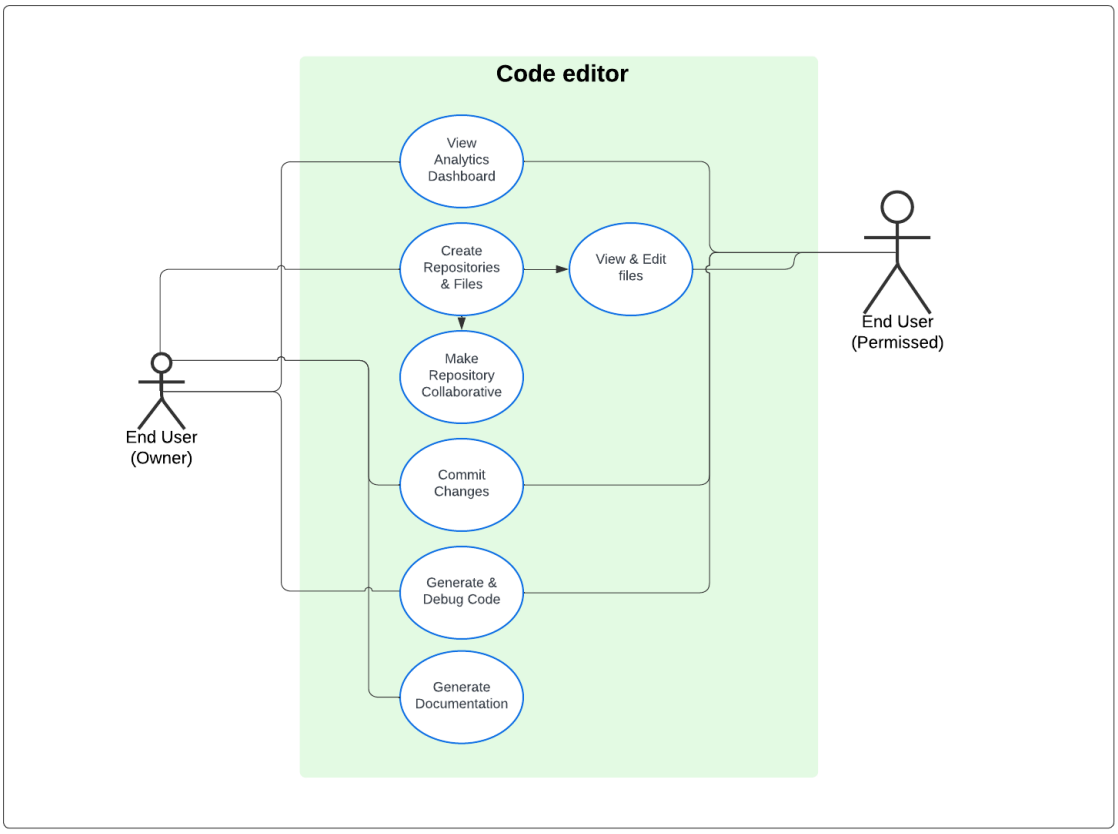

Figure 3.6: Use Case Diagram of the Proposed System

The core use cases include:

- **Create Repositories & Files**: Allows the user to initiate new projects by creating repositories and files.
- **View & Edit Files**: Enables the user to interact with the codebase, making necessary modifications.
- **Make Repository Collaborative**: Facilitates the ability for multiple users to work on the same repository in real time.
- **Commit Changes**: Ensures that modifications are saved and versioned correctly.
- **Generate & Debug Code**: Provides tools for compiling, running, and debugging code.
- **Generate Documentation**: Automatically generates documentation for the project based on the codebase.

Chapter 4

Project Implementation

The project implementation phase involves translating the design and planned architecture into a functional system, incorporating key features and components to meet the project's objectives. During this phase, we focused on building the core functionalities of the system, including backend development, user interface creation, and real-time data processing. Emphasis was placed on ensuring seamless integration between different technologies and tools, optimizing performance, and maintaining system scalability. Code snippets and critical components were tested iteratively to ensure functionality, robustness, and security. The project also included automated documentation generation and analytics tracking, providing insights into user interactions and system performance. Overall, the implementation phase is the backbone of the project, where conceptual designs are transformed into a working solution ready for further testing and deployment.

4.1 Timeline Sem VII

As software development projects grow in complexity, the need for well-organized project management and clear milestones becomes critical. The timeline presented in this section highlights the progress of our project during Semester VII, tracking key milestones from the conceptualization phase to design, implementation, and testing. Each stage has been carefully planned to maintain efficiency, mitigate risks, and ensure alignment with project goals.

This project involves numerous tasks that demand close collaboration between team members and adherence to deadlines. The Gantt chart below visually represents the progress and scheduling of these tasks, illustrating how each activity contributes to the overall workflow. As seen in Figure 4.1, the timeline captures task dependencies, reflecting how one task's completion influences the next, and how any delays could affect the project as a whole. This is particularly important as modern development workflows often involve overlapping tasks, requiring careful coordination to avoid bottlenecks or inefficiencies.

Throughout the semester, the team has adhered to a structured project schedule, broken down into various phases, each addressing specific goals. Initial phases included defining project scope, researching methodologies, and laying out architectural foundations, while later stages focused on coding, integration, and testing. Regular review meetings and progress checks ensured that the project remained on track, allowing for adjustments in task prioritization where necessary.

In addition to mapping out task timelines, the Gantt chart (Figure 4.1) also incorporates task completion percentages, offering a clear view of project progress at a glance. The team faced challenges typical of large-scale development projects, such as the need to manage multi- ple concurrent tasks, ensure timely collaboration, and meet set deadlines for deliverables. Nonetheless, through strategic planning and regular updates, these challenges were met, allowing the team to progress smoothly through each phase of development.

Overall, this timeline serves as a comprehensive overview of the project's lifecycle during Semester VII, providing insight into how the team has managed the complexities of the project while adhering to deadlines and ensuring timely task completion.

Figure 4.1: Timeline of the Project Milestones

4.2 System Prototype

The following section showcases the system prototype, which represents 25% of the total project implementation. This phase demonstrates the initial build of the core functionalities, including real-time collaborative editing, automated document generation, and analytics tracking. The prototypes detailed below serve as the foundation for the further development of the system's full features, providing insight into how the system will evolve in subsequent phases.

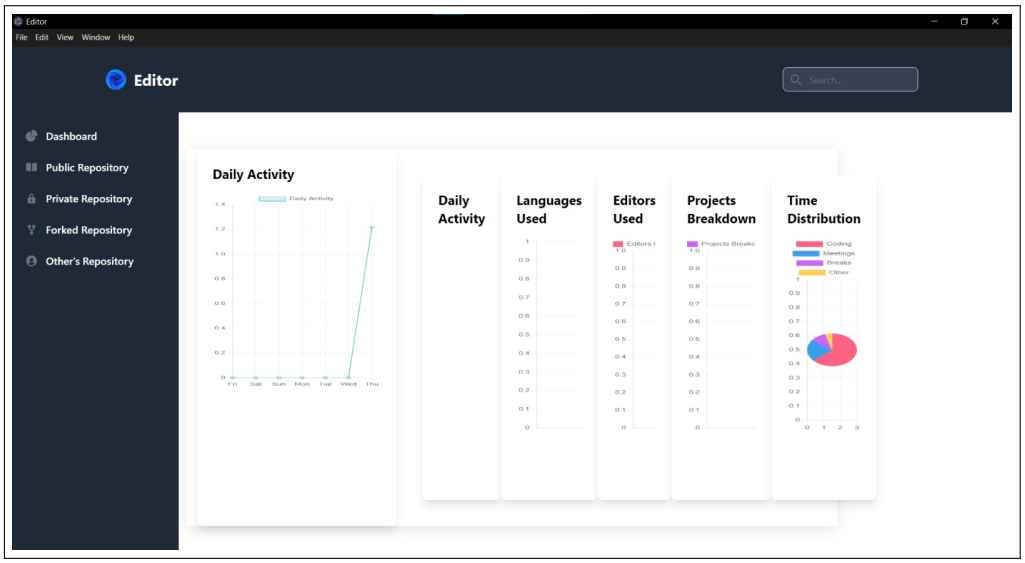

Figure 4.2: User behavioral analytics Dashboard

The figure 4.2 shown represents an analytics dashboard from a code editor interface. It provides a detailed overview of various aspects of a developer's activity and project management. On the left, the Daily Activity graph presents a visual representation of the user's coding or project involvement over a set period, showing a spike or increase on the most recent day. The Languages Used section outlines the programming languages employed by the user, though this particular dashboard indicates minimal variation, likely due to limited or singular language usage. Additionally, the Editors Used graph highlights the variety of code editors utilized during the development process, reflecting tool preferences. The Projects Breakdown section focuses on how different repositories—public, private, forked, or others—are managed, while the Time Distribution pie chart provides insight into how time is allocated among key tasks, such as coding, meetings, breaks, and other activities. This dashboard serves as a useful tool for tracking and optimizing development efforts in a collaborative or individual project environment.

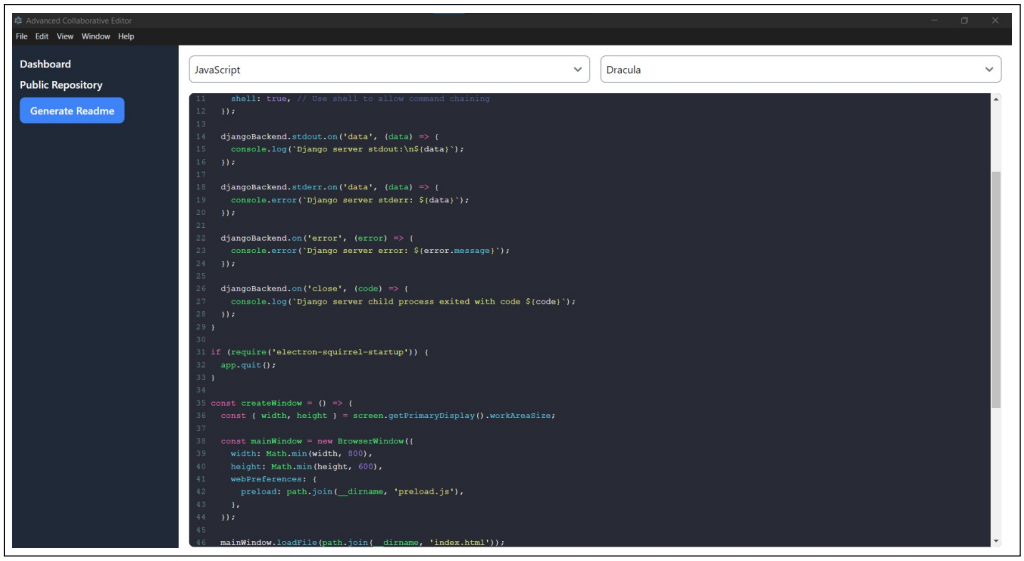

Figure 4.3: Collbaorative code editor

The figure 4.3 shows a collaborative code editor displaying JavaScript code, focused on integrating a Django backend with an Electron-based desktop application. The code handles server events such as stdout, stderr, and errors, logging outputs and messages to the console for debugging. It also listens for the electron-squirrel-startup event, exiting the app to prevent multiple instances during startup. The code includes logic for creating a new Electron BrowserWindow, defining its size and loading necessary files like preload.js and index.html, which likely form the user interface of the application. This code snippet demonstrates the seamless management of both server-side processes and the frontend interface, emphasizing real-time event handling and system responsiveness.

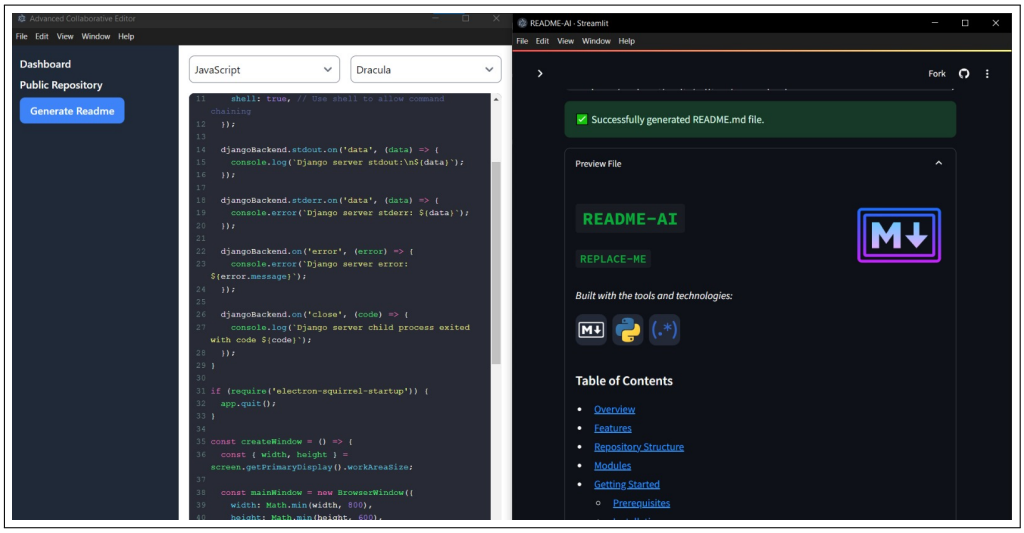

Figure 4.4: Document Generation

The figure 4.4 shows a dual-pane interface of a collaborative code editor. On the left, it displays a JavaScript code snippet managing server-side events for a Django backend in an Electron-based application, with event listeners for server output, error handling, and window creation. On the right, the successful generation of a README file is shown using

an automated tool, titled README-AI, with a table of contents featuring sections like Overview, Features, and Getting Started, and icons for technologies such as Markdown and Python. This setup demonstrates the seamless integration of coding and real-time documentation generation, streamlining both development and project organization.

Chapter 5

Summary

The Collaborative Code Editor project successfully tackled the core challenges in distributed software development by delivering a suite of advanced features focused on enhancing real-time collaboration, providing intelligent coding support, and improving overall workflow efficiency. Key features include real-time multi-user editing facilitated through WebSockets, operational transformation (OT), and conflict-free replicated data types (CRDT) algorithms to ensure seamless collaboration across multiple users. The system also integrates repository-level code completion powered by large language models (LLMs) like OpenAI Codex and CodeBERT, allowing for intelligent code suggestions that consider the entire project context. Automated code analysis capabilities were implemented to detect coding errors, ensure code quality, and generate inline documentation, helping developers maintain clean, efficient codebases. Additionally, a team performance dashboard was developed to track and optimize workflow, offering insights into team productivity and bottlenecks, ultimately leading to better decision-making and task management. The technical stack includes Python, JavaScript, Electron JS, FastAPI for the backend, PostgreSQL as the database, and Redis for message brokering, ensuring robust performance and scalability. By integrating advanced tools like Codex and CodeBERT for intelligent code completion, the system not only reduces coding errors but also accelerates development cycles through features like smart commits and semantic code search, allowing for more efficient navigation and management of the codebase. In conclusion, the project significantly improves collaboration among distributed teams, offers smarter and more context-aware code suggestions, streamlines workflows, and enhances team performance management. This results in higher productivity, better code quality, and a more organized and efficient approach to managing large codebases.

Bibliography

[1] Mahdi Jaberzadeh Ansari. An evaluation on automated technical documentation generator tools. *The University of Calgary*, 2022.

[2] Fabio Calefato, Giovanna Castellano, and Veronica Rossano. A revision control system for image editing in collaborative multimedia design. In *2018 22nd International Conference Information Visualisation (IV)*, pages 512–517. IEEE, 2018.

[3] Hongfei Fan, Kun Li, Xiangzhen Li, Tianyou Song, Wenzhe Zhang, Yang Shi, and Bowen Du. Covscode: a novel real-time collaborative programming environment for lightweight ide. *Applied Sciences*, 9(21):4642, 2019.

[4] Hongfei Fan, Hongming Zhu, Qin Liu, Yang Shi, and Chengzheng Sun. Shared-locking for semantic conflict prevention in real-time collaborative programming. In *2017 IEEE 21st International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 174–179. IEEE, 2017.

[5] Siyuan Jiang and Collin McMillan. Towards automatic generation of short summaries of commits. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, pages 320–323. IEEE, 2017.

[6] Yuta Koreeda, Terufumi Morishita, Osamu Imaichi, and Yasuhiro Sogawa. Larch: Large language model-based automatic readme creation with heuristics. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 5066–5070, 2023.

[7] Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331, 2023.

[8] Khushwant Virdi, Anup Lal Yadav, Azhar Ashraf Gadoo, and Navjot Singh Talwandi. Collaborative code editors - enabling real-time multi-user coding and knowledge sharing. In *2023 3rd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, pages 614–619, 2023.

[9] Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696*, 2017.

[10] Yury Zemlyanskiy, Michiel de Jong, Joshua Ainslie, Panupong Pasupat, Peter Shaw, Linlu Qiu, Sumit Sanghai, and Fei Sha. Generate-and-retrieve: use your predictions to improve retrieval for semantic parsing. *arXiv preprint arXiv:2209.14899*, 2022.

[11] Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. Repocoder: Repository-level code completion through iterative retrieval and generation, 2023.

[12] Shuyan Zhou, Uri Alon, Frank F Xu, Zhiruo Wang, Zhengbao Jiang, and Graham Neubig. Docprompting: Generating code by retrieving the docs. *arXiv preprint arXiv:2207.05987*, 2022.