Grenze International Journal of Engineering and Technology, January Issue

CodeIt: A Real-Time Collaborative Code Editor for Multi-user Programming

Rohan Wagh[1], Rohan Agrawal[2], Rohit Jagtap[3], Aniruddha Pande[4] and Ranjana Badre[5]
[1-5]School of Computer Engineering, MIT Academy of Engineering, Pune, 412105, Maharashtra, India.
Email: rohan.wagh@mitaoe.ac.in, rohan.agrawal@mitaoe.ac.in, rohit.jagtap@mitaoe.ac.in, aniruddha.pande@mitaoe.ac.in, rrbadre@mitaoe.ac.in

***Abstract*— Real-time collaborative code editors have become essential for remote software development, online education, and distributed teamwork. Existing platforms such as Cloud9 IDE, Saros, Collabode, Codeshare.io, and JSFiddle introduced early collaboration features but suffer from scalability issues, limited multi-file management, and lack of integrated execution or AI support. Advanced editors like Replit and VS Code Live Share address some of these challenges but often require heavy setup or fail to unify execution, communication, and intelligent coding assistance in a lightweight browser-based system. Research on concurrency control mechanisms such as Operational Transformation (OT) and Conflict-Free Replicated Data Types (CRDTs) has improved synchronization and consistency, yet their complexity limits adoption in practical implementations. To address these gaps, this paper presents CodeIt, a web-based collaborative coding platform that integrates synchronized multi file editing, secure multi-language execution via Piston API, AI-assisted code generation, real-time chat, presence indicators, and a collaborative whiteboard. Built with React, TypeScript, Node.js, Express, and Socket.IO, the system ensures low-latency synchronization, scalability, and reliable updates through a CI/CD pipeline. Performance evaluation demonstrated stable responsiveness under concurrent usage, validating its robustness. By unifying editing, execution, and intelligent assistance, CodeIt advances collaborative programming for education, coding interviews, hackathons, and distributed teams.**

***Index Terms*— Collaborative code editor, Real-time collaboration, Web-based programming, Socket.IO, Code execution, AI assisted coding, CI/CD pipeline.**

I. INTRODUCTION

Collaborative programming platforms are evolving rapidly, particularly in domains such as education, remote software development, and industry-level teamwork [10]. Real-time editors form the foundation of these systems, enabling developers to simultaneously contribute to shared projects without relying solely on version control or screen sharing.

However, designing collaborative editors introduces significant challenges, including maintaining low-latency synchronization, handling concurrent edits, ensuring scalability under heavy load, and integrating secure execution environments [2], [4]. To overcome these challenges, platforms must be lightweight, extensible, and capable of sustaining stable performance in diverse usage conditions.

Grenze ID: 01.GIJET.12.1.122

From the existing body of work, it is evident that while numerous collaborative editors exist, there remains a need for a comprehensive, browser-based, multi-user platform that integrates real-time editing, execution, communication, and AI powered assistance while remaining scalable under concurrent usage [1], [5], [6]. In this work, we propose CodeIt, a real-time collaborative coding platform designed to address these challenges. The system provides synchronized code editing across multiple files, live code execution through secure APIs, AI-powered code suggestions, real-time chat, presence indicators, and collaborative drawing tools. By integrating a CI/CD pipeline for deployment and ensuring extensibility, the proposed editor offers a unified environment that bridges the gap between lightweight online editors and full-fledged development platforms.

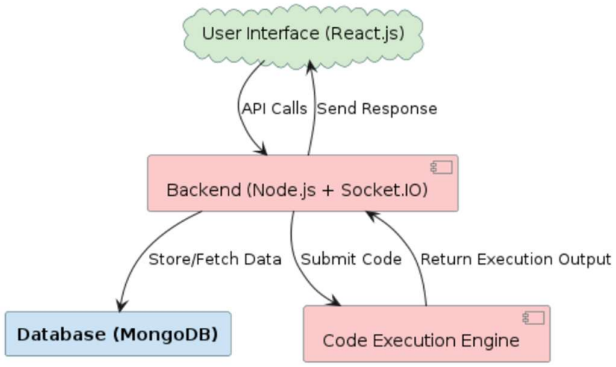

Fig. 1. Block Diagram.

II. Related Work

The development of collaborative programming tools has been driven by web-based code editors that address the limitations of traditional offline IDEs. Early platforms such as Cloud9 IDE, Saros, and Collabode demonstrated the feasibility of real-time collaboration in cloud environments [5] but often faced scalability challenges and lacked integrated execution support. Lightweight tools such as Codeshare.io and JSFiddle enabled quick collaboration but were generally limited to single-file editing and minimal project management.

A key challenge in real-time collaborative editors is maintaining consistency and responsiveness under concurrent access. Studies on large-scale systems have shown that server-side synchronization mechanisms can degrade with increasing numbers of simultaneous users, resulting in latency and inconsistencies [2]. To mitigate this, concurrency control mechanisms have been extensively studied. Operational Transformation (OT) is widely adopted due to its effectiveness in preserving user intent through server-coordinated transformation of concurrent edits [4]. Although decentralized approaches such as CRDTs have been explored, their complexity often limits their use in lightweight, browser-based editors [10].

Beyond synchronization, effective collaboration also depends on usability and developer awareness. Tools such as Spike provide visual feedback on fine-grained code changes, improving collaborators' understanding of ongoing activity [3]. Support for multiple programming languages is another essential requirement, with machine-learning-based approaches proposed to enable automatic language identification [8]. However, most existing tools focus on isolated aspects of collaboration. For example, CodeR supports real-time editing but lacks integrated execution support [6], while platforms such as Replit and VS Code Live Share offer strong collaboration features but do not fully unify editing, execution, and communication in a lightweight browser-based environment. These limitations motivate the need for a unified collaborative coding platform, as proposed in this work.

III. Proposed Methodology

The development of CodeIt is guided by a modular and layered methodology, ensuring extensibility, scalability, and robustness. The system is conceived as a browser-native platform that unifies real-time collaborative editing, secure code execution, and intelligent assistance in a single environment. It follows a client-server architecture, where the frontend handles user interaction and visualization, and the backend manages synchronization, authentication, and execution services. The overall system design is illustrated in Figure 2.

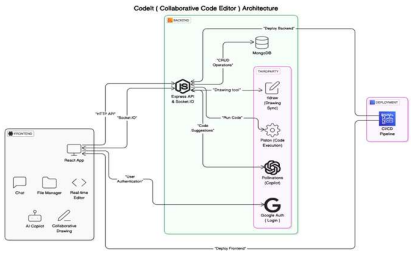

Fig. 2. System Architecture of CodeIt

A. Client-Side Architecture and User Experience

At the client layer, the system leverages a modern technology stack to deliver a responsive and feature-rich user interface. The frontend is a Single-Page Application (SPA) built with React.js and TypeScript. React's component-based architecture is exceptionally well-suited for a complex UI, allowing for the encapsulation of features like the editor, chat, and file manager into reusable and maintainable components. TypeScript enforces strict static typing, which significantly reduces runtime errors and improves the long-term maintainability of the codebase.

The core of the user experience is the editor component, which is built upon the Monaco Editor—the engine that powers Visual Studio Code. This choice provides users with a professional-grade editing experience out-of-the-box, including advanced features such as syntax highlighting for numerous languages, auto-indentation, language-specific for matting, and real-time error detection. This ensures that the collaborative environment remains lightweight and browser based without sacrificing the quality of a desktop IDE.

B. Real-Time Communication and Synchronization

The communication backbone between the clients and the server is managed by Socket.IO, a library that establishes persistent, low-latency WebSocket connections. Socket.IO was chosen over native WebSocket's for its robust feature set, including automatic reconnection on network interruptions and fallback to HTTP long-polling, which ensures connectivity even in restrictive network environments. Its room-based broadcasting API is fundamental to the platform's architecture, allowing the server to efficiently emit events – such as code changes, file operations, and chat messages—only to the participants within a specific collaborative session. To maintain data consistency across all clients during concurrent editing, the system adopts a centralized model based on the principles of Operational Transformation (OT). OT is a proven synchronization algorithm for real-time collaborative editors that excels at preserving user intent. The process is as follows:

- When a user makes an edit, the client generates a compact delta representing the change and sends it to the server.
- The server, acting as the single source of truth, receives this operation and transforms it against any other concurrent operations it has already processed.
- The transformed operation is applied to the master document state on the server.
- Finally, the server broadcasts the transformed operation to all other clients in the session, who then apply it to their local document replicas.

This server-authoritative approach ensures that all clients converge to an identical state, effectively preventing conflicts and maintaining a consistent, shared codebase even under heavy concurrent usage.

C. Backend Architecture and Data Management

The backend is developed using Node.js with the Express.js framework, a combination chosen for its performance in handling I/O-intensive workloads. Node.js's non-blocking, event driven architecture is highly efficient at managing many simultaneous WebSocket connections, which is the primary task of the server. Express.js provides a minimalist and flexible foundation for building the application's API end points and managing middleware for tasks like authentication and request handling.

Core responsibilities of the backend include:

- Session Management: Handling the lifecycle of collaborative rooms, including creation, joining and user authentication.
- Synchronization Logic: Acting as the central hub for the OT algorithm, processing and broadcasting all real-time events.

- Persistence: Managing the storage and retrieval of project and user data.

For persistent storage, MongoDB is employed. As a NoSQL database, its flexible, document-based model is ideal for storing the semi-structured data of the application, such as user profiles, project file trees, and chat histories. This schema flexibility allows for rapid iteration and development without the constraints of rigid SQL migrations, making it highly suitable for a real-time, evolving application.

D. Secure Multi-Language Code Execution

A critical feature of CodeIt is its ability to securely execute user-submitted code across multiple languages. Executing arbitrary code directly on the server poses significant security risks, including infinite loops, resource exhaustion, and malicious attacks. To mitigate these risks, the system offloads all execution tasks to the Piston API, a third-party service that runs code in secure, sandboxed environments.

The execution workflow is designed for security and isolation:

- The user initiates an execution request from the frontend, sending the source code, selected language, and any standard input to the CodeIt backend.
- The backend server acts as a secure proxy, constructing a 'POST' request to the Piston API's '/api/v2/execute' endpoint. This request payload includes the language, version, and an array of files containing the code to be executed.
- The Piston service provisions an isolated, ephemeral Docker container for the specified language runtime, executes the code within this sandboxed environment, and captures the output.
- Piston returns the execution results, including 'stdout', 'stderr', and the exit code, in a structured JSON response to the CodeIt backend.
- The backend streams these results back to the client in real time via the WebSocket connection, where they are displayed in the UI.

This methodology ensures strict separation between user code and the host environment, while resource limits and execution timeouts imposed by the Piston API safeguard against denial-of-service risks. The system currently supports widely used languages such as Python, Java, and C++, with a modular design that allows for easy extension to other languages.

E. DevOps and Continuous Deployment

To ensure scalability and maintainability, the project adopts modern DevOps practices. A continuous integration and deployment (CI/CD) pipeline are implemented using GitHub Actions. As illustrated in Figure 3, every code commit to the repository automatically triggers a series of workflows that build, test, and containerize the application. This automation validates the functionality and security of new changes before they are deployed, ensuring a high degree of reliability.

The containerized application is deployed on cloud infrastructure, enabling dynamic scaling based on user workload. This automated pipeline streamlines the release process, facilitates continuous improvements, and ensures high availability, making the system suitable for academic, professional, and competitive environments.

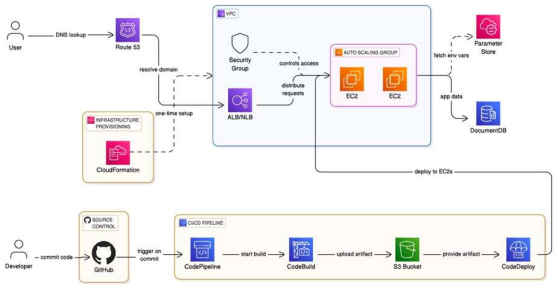

Fig. 3. CI/CD Pipeline for CodeIt

IV. Implementation and Results

A. System Implementation

The proposed system, CodeIt, was fully implemented according to the methodology outlined in Section III. The frontend was developed as a Single-Page Application using React and TypeScript, with the Monaco Editor serving as the core component for a professional-grade editing experience. The backend was deployed using a

Node.js and Express.js server, with MongoDB providing persistent storage for user sessions and project data. Real-time, bidirectional communication was managed through Socket.IO, and the Operational Transformation (OT) algorithm was implemented on the server to ensure conflict-free synchronization of concurrent edits. Secure, multi-language code execution was facilitated by integrating the Piston API, which processes code within isolated Docker containers.

The entire application was deployed via a CI/CD pipeline using GitHub Actions and Render, ensuring continuous and reliable updates.

B. User Interface and Functionalities

The user interface was designed to be intuitive and feature rich, providing a seamless collaborative experience. The key functionalities are presented below, with corresponding screen shots of the implemented system.

- Home Page and Room Management: The platform's entry point allows users to create a new collaborative room or join an existing one using a unique ID (Figure 4).
- Secure Room Access: To ensure privacy, the system implements an access control mechanism where the room owner must approve or deny join requests, which are communicated via email notifications (Figure 5).
- Real-Time Chat: An integrated chat module enables fluid, text-based communication among all participants within a session, as depicted in (Figure 6).
- AI-Powered Code Generation: An AI Copilot, powered by the Pollination AI API, provides intelligent code suggestions based on natural language prompts, allowing users to generate, copy, or insert code snippets directly into the editor (Figure 7).
- Collaborative Drawing Tool: A Tldraw-based white board is integrated into the workspace, supporting real time sketching, diagramming among collaborators (Figure 8).
- Integrated Code Execution: Users can execute their code directly within the editor and view the output in a dedicated panel, streamlining the development and debugging cycle (Figure 9).

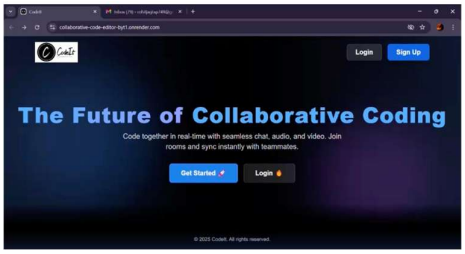

Fig. 4. Home Page

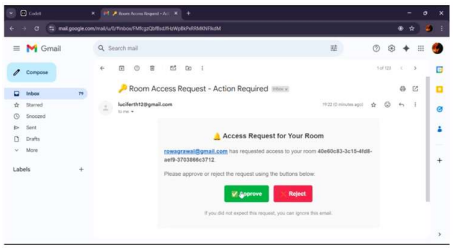

Fig. 5. Room Access Request via Email

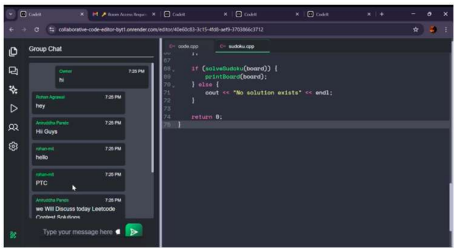

Fig. 6. Real-Time Chat Interface

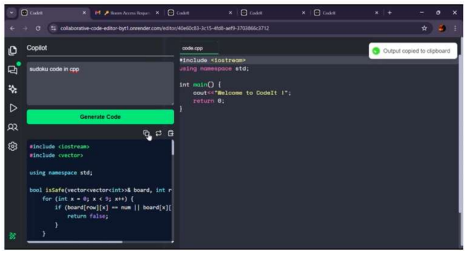

Fig. 7. AI-assisted Code Generation

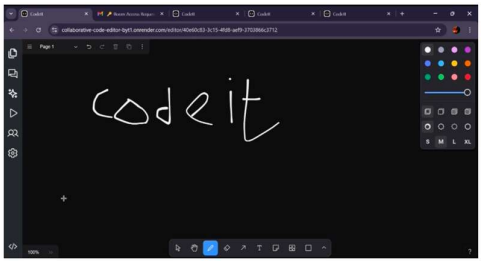

Fig. 8. Collaborative Drawing Tool

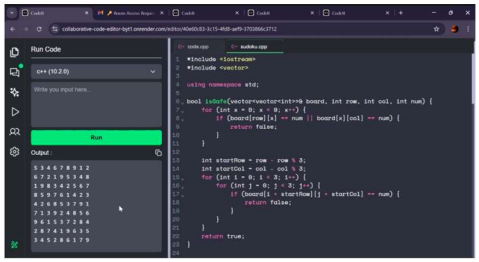

Fig. 9. Code Execution

C. Performance Evaluation

To validate the system's robustness and efficiency, a performance evaluation was conducted by simulating multiple concurrent users on a cloud-hosted instance. The primary objective was to measure the system's responsiveness and stability under varying loads.

- Experimental Setup and Parameters: The tests were performed on the deployed application, with simulated clients connecting to a single collaborative room and performing simultaneous editing and execution tasks. The key performance indicators (KPIs) and parameters were:
  - Concurrent Users: The number of simultaneous users was scaled from 2 to 50 to assess the impact on performance.
  - Supported Languages: Execution tests were conducted using Python, Java, and C++.
  - Synchronization Latency: Measured as the time delay between a character being typed by one user and appearing on the screens of other users.
  - Code Execution Time: Measured as the round-trip time from submitting a code snippet for execution to receiving the complete output from the Piston API.
- Results and Analysis: The experimental results, summarized in Table I, demonstrated stable and responsive performance under increasing concurrency. The real-time synchronization latency for collaborative editing remained well below the 100 ms threshold, which is considered the standard for a fluid user experience. This low latency is a direct result of the efficient WebSocket communication managed by Socket.IO and the lightweight delta-based updates.

The average execution time across the supported languages was consistently measured at approximately 1.2 seconds. This stability confirms the effectiveness of offloading code execution to the sandboxed Piston API, which prevents the main application server from becoming a bottleneck, even as the number of users increases.

TABLE I: EXPERIMENTAL RESULTS OF CODEIT

Concurrent Users	Latency (ms)	Execution Time (s)
5	40	1.1
10	55	1.2
15	65	1.2
20	75	1.3

Overall, the results confirm that the architecture of CodeIt is robust and maintains high responsiveness under moderate to-high concurrency. This validates its effectiveness for the intended use cases, such as interactive educational workshops, competitive hackathons, and collaborative development for distributed professional teams.

V. DISCUSSION

A. Comparison with Existing Tools

To contextualize the contributions of CodeIt, Table II provides a direct feature comparison against two leading collaborative development platforms: VS Live Share and Replit.

TABLE II: FEATURE COMPARISON OF COLLABORATIVE CODING PLATFORMS

Feature	**CodeIt**	**VS Live Share**	**Replit**
Primary Use Case	Integrated coding & learning	Extending local IDE	AI-driven app development
Real-time Editing	Yes	Yes	Yes
Shared Terminal	No (Output only)	Yes	Yes
Shared Debugging	No	Yes	Yes
Live Code Execution	Yes (via API)	Yes (Host's server)	Yes (Native)
AI Assistance	Yes (Code Generation)	Yes (via extensions)	Yes (Native Agent)
Visual Collaboration	Yes (Whiteboard)	No	No
Environment Setup	None (Browser-based)	Host IDE setup required	None (Browser- based)

As the comparison in Table II illustrates, established platforms excel in specific domains. VS Live Share is powerful for extending a developer's local, full-featured IDE for remote collaboration, offering deep integration with tools like shared debugging and terminals [10]. Replit provides a robust, all-in-one, browser-based environment with a strong focus on AI-driven development and native execution capabilities. However, both of these platforms, while feature-rich, lack an integrated tool for real-time visual brainstorming.

In contrast, the primary innovation of CodeIt is its unique synthesis of features. It combines the essential elements of real-time editing, secure remote execution, and AI assistance with a built-in collaborative whiteboard - a feature absent in the other platforms compared. This combination creates a unified cognitive space where developers can not only write and test code but also visualize architectural ideas and brainstorm solutions within the same environment. While other early platforms like Cloud9 IDE offered cloud-based editing [5], they lacked this level of seamless integration between coding, communication, and visual ideation. By unifying these traditionally dispersed functionalities, CodeIt offers a lightweight, browser-based solution optimized for hackathons, classroom use, and distributed professional teams.

B. Advantages of the Proposed System

The system integrates several innovative features that differentiate it from existing tools:

- AI-Assisted Coding: Provides intelligent code suggestions and generation capabilities, improving productivity and reducing boilerplate efforts.
- Collaborative Drawing Tool: Supports sketching of system designs, algorithms, and brainstorming ideas within the same environment, an element absent in most editors.
- Secure Execution: Executes code in sandboxed environments Piston API, ensuring isolation and security while supporting multiple programming languages.
- Integrated CI/CD Pipeline: Automates build, testing, and deployment, ensuring continuous integration and updates without manual intervention.
- Unified Collaboration: Combines editing, execution, chat, presence indicators, and visualization in a single lightweight browser interface. These features collectively position CodeIt as a versatile platform capable of serving academic, competitive, and professional collaboration contexts.

C. Limitations

While the system demonstrates strong performance under moderate concurrency, several limitations remain:

- Scalability: Current testing was limited to 50 concurrent users; larger deployments may require advanced clustering and distributed storage mechanisms.
- Network Dependency: The system is highly dependent on stable internet connectivity, which may limit usability in regions with unreliable bandwidth.
- Security Risks: Although Docker provides isolation, executing arbitrary user code introduces potential vulnerabilities that require constant monitoring, patching, and resource throttling.
- Feature Integration Overhead: Adding AI assistance and drawing tools increases resource utilization, which may impact performance under high-load conditions.

D. Summary

The discussion highlights that CodeIt bridges the gap between lightweight code-sharing tools and full-fledged integrated development environments. Its distinguishing features - AI support, real-time drawing, and secure execution enhance collaborative programming experiences, but further work is required to improve scalability, optimize resource usage, and strengthen security measures for large-scale deployment.

VI. CONCLUSION

In this work, a real-time collaborative code editor, CodeIt, was designed and developed to address the challenges of multi-user programming in distributed environments. The system integrates synchronized editing across multiple files, code execution through sandboxed APIs, and advanced collaboration features such as real-time chat, presence indicators, and notifications. To enhance developer productivity, the editor incorporates syntax highlighting, auto-language detection, customizable themes, and an AI-assisted coding module for intelligent suggestions and completions.

A collaborative drawing feature was further introduced to support brainstorming and diagram-based communication alongside code editing. The platform ensures scalability and reliability through a CI/CD pipeline, enabling continuous deployment and integration with cloud infrastructure. Performance evaluation demonstrated stable synchronization, low latency in real-time collaboration, and responsive execution across diverse use cases.

These results validate the potential of CodeIt as a unified platform that bridges the gap between lightweight web-based editors and full-fledged integrated development environments (IDEs).

REFERENCES

[1] M. Agrawal, J. Goyal, M. Goyal, P. Sukhija, J. Sharma, and D. F. Vinod, "Codexchange: Leaping into the future of AI-powered code editing," in 2024 International Conference on Computational Intelligence and Computing Applications (ICCICA), vol. 1, pp. 367–374, 2024. doi: 10.1109/ICCICA60014.2024.10585043. [2] Q.-V. Dang and C.-L. Ignat, "Performance of real-time collaborative editors at large scale: User perspective," in 2016 IFIP Networking Conference (IFIP Networking) and Workshops, pp. 548–553, 2016. doi: 10.1109/IFIPNetworking.2016.7497258.

[2] R. Escobar, J. P. S. Alcocer, H. Tarner, F. Beck, and A. Bergel, "Spike a code editor plugin highlighting fine-grained changes," in 2022 Working Conference on Software Visualization (VISSOFT), pp. 167–171, 2022. doi: 10.1109/VISSOFT55257.2022.00026.

[3] A. Imine, "Flexible concurrency control for real-time collaborative editors," in 2008 The 28th International Conference on Distributed Computing Systems Workshops, pp. 423–428, 2008. doi: 10.1109/ICDCS.Workshops.2008.91.

[4] W. Kimpan, T. Meebunrot, and B. Sricharoen, "Online code editor on private cloud computing," in 2013 International Computer Science and Engineering Conference (ICSEC), pp. 31–36, 2013. doi: 10.1109/IC SEC.2013.6694748.

[5] A. Kurniawan, C. Soesanto, and J. Wijaya, "CodeR: Real-time code editor application for collaborative programming," Procedia Computer Science, vol. 59, pp. 510–519, Dec. 2015. doi: 10.1016/j.procs.2015.07.531.

[6] M.-M. Mateas and C. Marsavina, "PIE: A general-purpose code editor focused on simplicity," in 2024 IEEE 18th International Symposium on Applied Computational Intelligence and Informatics (SACI), pp. 291 296, 2024. doi: 10.1109/SACI60582.2024.10619920.

[7] A. H. Odeh, M. Odeh, and N. Odeh, "Using multinomial naive Bayes machine learning method to classify, detect, and recognize programming language source code," in 2022 International Arab Conference on Information Technology (ACIT), pp. 1–5, 2022. doi: 10.1109/ACIT57182.2022.9994117.

[8] M. Rantala, J. Soini, and T. Kilamo, "Gathering useful programming data: Analysis and insights from real-time collaborative editing," in 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 229–234, 2015. doi: 10.1109/MIPRO.2015.7160270.

[9] K. Virdi, A. L. Yadav, A. Gadoo, and N. S. 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), pp. 614–619, Dec. 2023. doi: 10.1109/ICIMIA60377.2023.10426375.