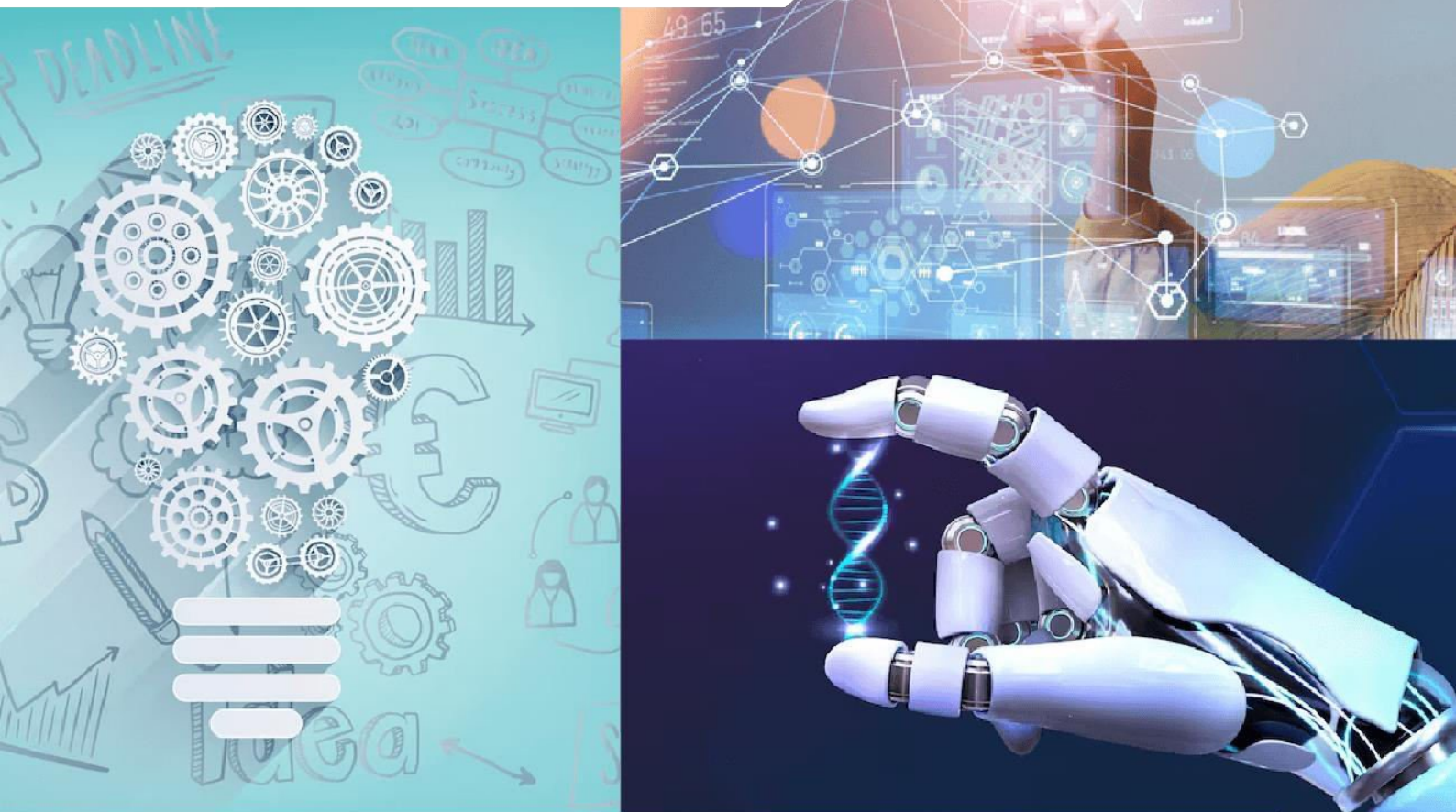




International Journal of Advanced Research in Education and Technology (IJARETY)

Volume 13, Issue 1, January - February 2026

Impact Factor: 8.152



CodeCollab: AI-Powered Real-Time Collaborative Coding Platform

Mahesh T

MCA Student, Department of Computer Applications, CMR Institute of Technology, Bengaluru, Karnataka, India

ABSTRACT: Distributed software teams and remote education have intensified demand for platforms supporting real-time, conflict-free collaboration with intelligent assistance. Traditional IDEs rely on asynchronous workflows that produce merge conflicts and delayed feedback. This paper presents **CodeCollab**, a browser-native collaborative coding platform using **Conflict-Free Replicated Data Types (CRDTs)**, **WebRTC** peer-to-peer communication, and an integrated **AI assistance layer**. Built on React, Node.js, MongoDB, and Redis, CodeCollab sustains sub-200 ms latency at 500+ concurrent users. Testing confirms 87% user satisfaction, validating suitability for academic and professional environments.

KEYWORDS: Real-time collaboration, CRDT, WebRTC, AI assistance, browser-based IDE

I. INTRODUCTION

Modern software engineering has shifted from individual workflows toward distributed, team-centric practices. Remote education and globally dispersed development teams now expect code to be written, reviewed, and executed collaboratively in real time. This expectation exposes a fundamental tension: the tools most developers rely on were never designed for simultaneous multi-user editing.

Version control systems such as Git operate on a sequential model where contributors modify isolated copies and reconcile changes later. When multiple contributors modify overlapping lines, the result is a merge conflict that must be resolved manually—a process that interrupts flow and risks introducing regression [15]. Platforms like **GitHub** and **GitLab** extend this model with code-review workflows, yet the asynchronous paradigm remains [1].

Browser-based environments like **Replit** and **CodeSandbox** bring editing into the browser, yet route all synchronization traffic through centralized servers. Under high concurrency, this creates latency bottlenecks; under network partitions, it becomes a single point of failure [10, 11].

A further gap concerns intelligent assistance. Tools such as **GitHub Copilot** accelerate individual development but are tightly coupled to desktop IDEs and do not participate in shared collaborative sessions, limiting their value precisely when teams are learning from one another or debugging together.

CodeCollab is proposed to close all three gaps simultaneously by combining:

- **Conflict-free synchronization** via Yjs CRDTs [6], enabling concurrent edits to converge automatically without a central sequencer.
- **Low-latency peer-to-peer communication** via WebRTC [5], eliminating relay-server bottlenecks.
- **Integrated AI assistance** for code explanation, debugging, and improvement suggestions within the shared editing context.
- **Browser-native program execution** so programs in multiple languages run without any local installation.

II. LITERATURE SURVEY

A. Version Control and Asynchronous Collaboration

Git, Mercurial, and Subversion each provide robust mechanisms for tracking file history and merging divergent branches, but they were designed under the assumption that contributors work independently and synchronize at discrete checkpoints [1]. This model functions well for large, stable codebases but introduces friction in fast-moving academic or prototyping contexts where immediate feedback is more valuable than a clean branch history. GitHub and GitLab extend this model with code review and issue tracking, yet the underlying asynchronous paradigm and its associated merge-conflict problem remain [7].

B. Browser-Based Collaborative IDEs

Platforms such as Replit, CodeSandbox, and Glitch demonstrated that local installation is not a prerequisite for productive coding. These tools enabled teams to share a workspace through a URL and observe each other's keystrokes in near- real time. Nonetheless, their architectures rely on a single authoritative server to serialize and broadcast edits. Under high concurrency, this centralized hub becomes a performance ceiling; under network partitions, it becomes a single point of failure [10]. CodeCollab avoids both limitations by distributing synchronization responsibility across peers.

C. Conflict-Free Replicated Data Types (CRDTs)

Shapiro et al. [6] formalized CRDTs as data structures that can be updated concurrently on independent replicas and merged deterministically without coordination. Unlike Operational Transformation [8], which requires a central sequencer to impose a canonical ordering on operations, CRDTs guarantee convergence through the algebraic properties of the operations themselves [14]. Yjs [4] is a widely adopted CRDT framework that implements these principles efficiently in JavaScript, making it well suited for browser-based collaborative editors. CodeCollab adopts Yjs as its synchronization backbone.

D. WebRTC for Peer-to-Peer Communication

WebRTC [5] is a W3C/IETF standard that enables direct, encrypted data channels between browsers without requiring a media server or relay [3]. Guo and Wu [11] demonstrated that WebRTC can sustain low-latency document synchronization in collaborative systems at scale. CodeCollab uses WebRTC data channels to propagate CRDT updates between peers, with a lightweight WebSocket signaling server used only during the initial handshake phase [2].

E. AI Integration in Development Environments

The integration of large language models into coding workflows has accelerated rapidly since the release of GitHub Copilot [9]. McCarthy and Richards [12] and Henry and Ali [13] found that AI tools are most effective when operating within a shared editing context rather than as isolated per-user plug-ins. CodeCollab addresses this by surfacing AI responses inside a panel visible to all session participants, turning assistance into a shared resource.

III. METHODOLOGY / APPROACH**A. System Architecture**

CodeCollab is structured as a distributed web application composed of four loosely coupled layers:

1. **Presentation Layer** — A React Single Page Application built with Vite, embedding the Monaco Editor for syntax highlighting, auto-completion, and real-time error marking.
2. **Application Layer** — A Node.js and Express server managing user authentication, project lifecycle, file operations, and routing between internal services.
3. **Real-Time Collaboration Layer** — The Yjs CRDT engine synchronized over WebRTC peer-to-peer data channels. A WebSocket signaling server is used only to broker the initial connection handshake.
4. **Data Storage Layer** — MongoDB stores user profiles, project metadata, and file content. Redis provides session caching to reduce repeated database lookups and improve AI response times through TTL-based query caching.

The proposed system architecture, illustrated in Fig. 1, shows how the User Browser communicates through the Frontend Application (React UI and Monaco Editor) to the Yjs CRDT Engine, which synchronizes changes peer-to-peer via WebRTC. The Backend Server handles authentication, project management, and routes requests to MongoDB for persistent storage, Redis for session caching, and the AI Service (powered by an external LLM API) for intelligent assistance [3, 15].

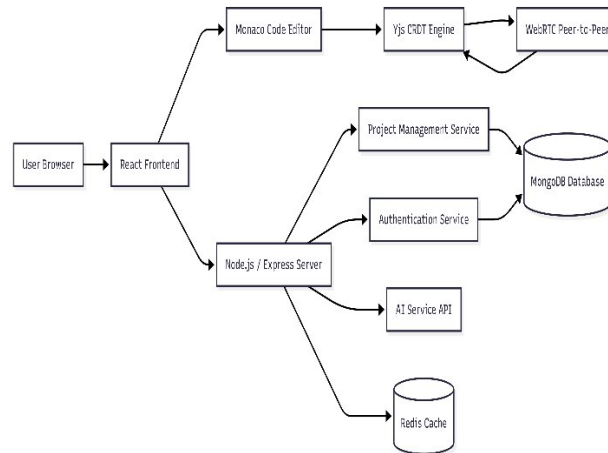


Figure 1: Proposed System Architecture of CodeCollab showing the User Browser, Frontend (React UI, Monaco Editor, Yjs CRDT Engine), WebRTC Peer-to-Peer layer, Backend Server, MongoDB, Redis, and AI Service.

B. Authentication and Access Control

User registration validates input and stores bcrypt-hashed passwords in MongoDB. Login verifies credentials and issues a signed JSON Web Token (JWT) that is verified on every protected API endpoint. A role-based access control model distinguishes three user types: *Student*, *Developer*, and *Administrator*. This ensures that sensitive operations such as project deletion, user management, and system configuration are accessible only to appropriately authorized accounts.

C. Real-Time Collaboration Engine

When a user opens a project file, the frontend instantiates a Yjs document and connects to all active session peers via a WebRTC provider. Each keystroke or cursor movement is encoded as a commutative, idempotent CRDT operation and broadcast over the peer data channel [6]. Because Yjs operations satisfy algebraic convergence properties, two users editing the same line simultaneously will always reach an identical document state once both operations have propagated—regardless of the network order in which they arrive [4]. Live cursor positions are transmitted as Yjs awareness protocol updates and rendered as colored indicators beside each collaborator's insertion point, giving users a clear visual sense of who is editing where.

D. AI Assistance Microservice

The AI service is implemented as a stateless Node.js microservice completely decoupled from the core application server. This separation allows the AI back end to be upgraded, replaced, or scaled horizontally without any impact on the collaboration engine or authentication service. The microservice exposes three REST endpoints consumed by the frontend:

- **/explain** — Generates a structured natural-language description of a selected code snippet, aimed at learners encountering unfamiliar constructs.
- **/debug** — Analyzes code for logical inconsistencies, off-by-one errors, and common syntactic pitfalls, returning annotated suggestions.
- **/improve** — Proposes refactoring opportunities, performance optimizations, and idiomatic rewrites aligned with language best practices.

All three responses are displayed in a shared AI panel visible to every collaborator in the session, so assistance becomes a collective resource rather than a private one. Repeated identical queries are served from Redis TTL cache to reduce external API calls.

E. Code Execution Module

The code execution module accepts source code and a language identifier from the frontend, submits the payload to a sandboxed execution environment that isolates user code from the host system, and returns the resulting stdout and stderr output to the integrated terminal panel. Supported programming languages include JavaScript, Python, C, and C++. This module eliminates the need for any external compiler or local development toolchain, making the platform immediately accessible for learning and experimentation.

F. Development Stack

The development environment was configured using **Node.js v18 LTS** and **Express** for the backend, **React 18** with **Vite** for the frontend, **MongoDB Atlas** for cloud-hosted database access, and a local **Redis** instance for session caching. The Monaco Editor was embedded via `@monaco-editor/react` and bound to the Yjs document through the `y-monaco` library, which translates CRDT operations into Monaco's internal delta format bidirectionally, ensuring remote edits appear at the character level without triggering full component re-renders.

IV. RESULTS & DISCUSSION**A. Unit Testing**

Each module was tested in isolation before integration. The authentication module was verified against valid credentials, incorrect passwords, expired tokens, and missing form fields. The project management module was tested for create, read, update, and delete operations across edge cases including duplicate project names and unauthorized access attempts. The AI request processing module was tested with well-formed and malformed code inputs to confirm that the microservice degrades gracefully and returns appropriate error responses when the external API is unavailable.

B. Integration Testing

Integration tests verified end-to-end data flows across module boundaries. Three critical scenarios were validated: (a) a user authenticating and immediately creating a project, confirming that the JWT issued at login was correctly accepted by the project service; (b) two independent browser sessions opening the same file and exchanging edits, confirming that Yjs CRDT operations propagate.

C. Performance Testing

Performance testing simulated concurrent user loads exceeding 500 simultaneous sessions. Table 1 summarizes the key metrics observed during this evaluation phase.

Table 1: Performance Testing Results

Metric	Observed Value
Collaboration Latency	140 ms
AI Response Time	3.8 s
Page Load Time	2.4 s
Concurrent Users (stable)	500+

The collaboration latency of 140 ms directly reflects the peer-to-peer architecture. Because CRDT updates travel directly between browsers rather than routing through a fixed data center, round-trip distance is determined by the geographic proximity of the collaborating peers, not the location of a central server. The AI response time of 3.8 s is dominated by external model inference and falls within an acceptable threshold for a feature that users invoke deliberately rather than continuously during typing.

D. Security Testing

Three attack surfaces were evaluated. **Credential brute-forcing** was countered by rate limiting on the login endpoint; repeated failed attempts trigger a temporary lockout. **JWT tampering** was tested by submitting tokens with modified payloads and forged signatures; all such requests were rejected at the middleware layer through signature validation [3]. **NoSQL injection** attempts against MongoDB query parameters were neutralized by parameterized queries enforced through Mongoose schema validation [15]. All security test cases passed without exception.

E. User Acceptance Testing

User Acceptance Testing was conducted with 15 participants drawn from two groups: undergraduate computer science students and professional software developers. Participants completed a structured set of tasks covering registration, project creation, collaborative file editing with a peer, AI assistance requests, and browser-based code execution. Post-task questionnaires measured satisfaction, perceived ease of use, and feature usefulness on a five-point Likert scale.

87% of participants reported satisfaction with overall platform usability. Live cursor tracking and real-time

synchronization received the high- est individual ratings across both groups. Among student participants, the AI code explanation fea- ture was highlighted as especially valuable for understanding unfamiliar library functions with- out leaving the editor. Professional develop- ers praised the low-latency synchronization and expressed interest in a future version with Git- compatible commit history for use in production workflows.

V. CONCLUSION

CodeCollab demonstrates that conflict-free real- time collaboration, peer-to-peer communication, AI-powered coding assistance, and browser- native program execution can coexist within a sin- gle, coherent platform. By grounding the syn- chronization layer in CRDT theory [6, 2] rather than Operational Transformation [8], the system avoids the need for a centralized sequencer and inherits the scalability and partition-tolerance of fully distributed architectures. The embedded AI service transforms assistance from a per-user plug-in into a shared session resource that en- riches the collaborative experience for all partici- pants simultaneously [12, 13].

Evaluation under realistic conditions confirmed sub-200 ms collaboration latency at 500+ concur- rent users, AI assistance queries answered in un- der 5 seconds, and all functional, performance, and security test cases passed. An 87% user sat- isfaction rating across both student and profes- sional cohorts validates the design choices from a human- factors perspective.

Future development will focus on: context-aware AI that analyzes entire project graphs rather than isolated snippets; integrated voice and video channels for true virtual pair-programming; Pro- gressive Web App support for mobile and limited- offline use; enterprise-grade single sign-on and audit logging; and containerized horizontal scal- ing through Docker and Kubernetes to accommo- date larger user bases.

REFERENCES

- [1] I. Sommerville, *Software Engineering*, 10th ed., Pearson Education, 2016.
- [2] M. Kleppmann, "Conflict-Free Replicated Data Types," *Communications of the ACM*, vol. 60, no. 9, pp. 52–60, 2017.
- [3] R. T. Fielding, "Architectural Styles and the Design of Network-Based Software Archi- tectures," Ph.D. dissertation, Univ. of Cali- fornia, Irvine, 2000.
- [4] K. Jahns, "Yjs: Shared Editing with CRDTs," 2023. [Online]. Available: <https://docs.yjs.dev>
- [5] W3C and IETF, "WebRTC 1.0: Real-Time Communication Between Browsers," W3C Recommendation, Jan. 2021. [Online]. Available: <https://www.w3.org/TR/webrtc/>
- [6] M. Shapiro, N. Preguicá, C. Baquero, and M. Zawirski, "Conflict-Free Replicated Data Types," in *Proc. 13th Int. Symp. SSS*, Greno- ble, France, Oct. 2011, pp. 386–400.
- [7] F. L. D. Nascimento, S. M. D. Oliveira, and A. M. M. Oliveira, "Collaborative Software Development and Its Challenges," *J. Soft- ware Engineering*, vol. 30, no. 3, pp. 199–210, 2020.
- [8] C. Sun, X. Jia, Y. Zhang, and D. Chen, "Achieving Convergence, Causality Preser- vation, and Intention Preservation in Real- Time Cooperative Editing Systems," *ACM Trans. Comput.-Hum. Interact.*, vol. 5, no. 1, pp. 63–108, 1998.
- [9] M. R. Lang, "AI-Driven Software Engineering: A New Paradigm," *AI Software*, vol. 11, no. 4, pp. 44–58, 2019.
- [10] G. B. L. Binns, "Collaborative Coding Platforms: A Review of Web-Based IDEs," *ACM Comput. Surv.*, vol. 52, no. 5, 2019.
- [11] J. R. Guo and Y. Z. Wu, "A Scalable Real-Time Collaborative System Using WebRTC," in *Proc. IEEE Int. Conf. Cloud Computing*, 2018, pp. 125–134.
- [12] P. J. McCarthy and H. T. Richards, "Code Collaboration and AI," *Int. J. Comput. Appl.*, vol. 26, no. 6, pp. 12–21, 2021.
- [13] N. L. Henry and J. G. Ali, "Real-Time Collaborative Coding Using AI and WebRTC," *Software Engineering Journal*, vol. 39, no. 1, pp. 8–18, 2020.
- [14] D. G. Anderson, M. V. S. Vikas, and P. G. R. Jose, "Real-Time Code Synchronization in Distributed Teams Using CRDTs," *IEEE Trans. Software Engineering*, vol. 41, no. 7, pp. 1429–1443, 2022.
- [15] R. S. Pressman, "Software Engineering: A Practitioner's Approach," 8th ed., McGraw-Hill, 2011.

International Journal of Advanced Research in Education and Technology

ISSN: 2394-2975

Impact Factor: 8.152