International Journal of Scientific Research and Engineering Development-– Volume 9 Issue 2, Mar-Apr 2026
Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

AI-Powered Collaborative Code Editor Platform: Integrating Real-Time Collaboration, Artificial Intelligence, and Competitive Programming

Priyanka A. Chorey[1], Sarvesh Bijwe[2], Gayatri Awathe[3], Rasika Bhusari[4], Harshwardhan Vane[5]

[1](Department of Information Technology, Prof. Ram Meghe Institute of Technology & Research, Badnera, India Email: priyankachorey07@gmail.com)

[2](Department of Information Technology, Prof. Ram Meghe Institute of Technology & Research, Badnera, India Email: sbijwe8080@gmail.com)

[3](Department of Information Technology, Prof. Ram Meghe Institute of Technology & Research, Badnera, India Email: awathegayatri@gmail.com)

[4](Department of Information Technology, Prof. Ram Meghe Institute of Technology & Research, Badnera, India Email: rasikabhusari2@gmail.com)

[5](Department of Information Technology, Prof. Ram Meghe Institute of Technology & Research, Badnera, India Email: vaneharsh@gmail.com)

----------**********----------

Abstract:

The proliferation of remote software development and AI-assisted coding tools has created demand for integrated platforms that combine collaborative editing, intelligent assistance, and competitive programming. This paper presents the design, implementation, and evaluation of an AI-Powered Collaborative Code Editor Platform, a web-based application that unifies real-time multi-user code editing with AI-driven code assistance (powered by Google Gemini) and structured competitive programming features including timed battles, automated judging, and global leaderboards . The platform architecture leverages React with TypeScript for the frontend, Supabase (PostgreSQL, Authentication, Realtime WebSocket channels, Edge Functions) for backend services, Monaco Editor for the code editing experience, and Judge0 API for sandboxed code execution across seven programming languages. Row-Level Security policies and role-based access control ensure data protection. Experimental evaluation demonstrates sub-second code synchronization latency (~100ms), AI response first-token delivery under 1.5 seconds via streaming Server-Sent Events, and reliable code execution within acceptable time frames. A comparative analysis against existing platforms (VS Code Live Share, Replit, LeetCode, HackerRank) reveals that no current solution integrates all three paradigms within a single freely accessible web application, establishing this work's contribution to the field.

Keywords: Collaborative Code Editor, Real-Time Synchronization, Artificial Intelligence, Large Language Models, Competitive Programming, WebSocket, Monaco Editor, Supabase, Judge0, React.

----------**********----------

I. INTRODUCTION

The landscape of software development has undergone fundamental transformations driven by three concurrent trends: the global shift toward remote and distributed development workflows, the emergence of Large Language Models (LLMs) capable of understanding and generating code with remarkable proficiency, and the growing popularity

of competitive programming platforms for skill assessment and development [1].

Traditional Integrated Development Environments (IDEs) such as Visual Studio Code, IntelliJ IDEA, and Eclipse, while powerful for individual development, were not designed for seamless multi-user real-time collaboration. Conversely, collaborative editing platforms like Replit and CodeSandbox lack structured competitive programming features. Competitive platforms like LeetCode and HackerRank do not support real-time collaboration or integrated AI assistance [2], [3].

This paper addresses the identified gap by presenting the design and implementation of an AI-Powered Collaborative Code Editor Platform that integrates three previously siloed capabilities into a unified, web-based application:

Real-time collaborative code editing with live synchronization, cursor tracking, and presence indicators

AI-powered coding assistance including code explanation, automated debugging, optimization suggestions, and conversational AI support

Competitive programming with a curated problem library, timed code battles, automated test case evaluation, and global leaderboard

The remainder of this paper is organized as follows: Section II reviews related work across collaborative editing technologies, AI code assistance, and online judge systems. Section III details the system architecture and design. Section IV describes the implementation methodology. Section V presents experimental results and provides a detailed performance evaluation of the proposed system. Section VI concludes with contributions and future directions.

II. RELATABLE WORK

A. Collaborative Editing Technologies

Real-time collaborative editing is a well-studied distributed systems problem. Operational Transformation (OT), introduced by Ellis and Gibbs [4], resolves concurrent editing conflicts by transforming operations relative to each other. Google Docs and many early collaborative editors employ OT. Conflict-free Replicated Data Types (CRDTs), formalized by Shapiro et al. [5], offer a mathematically proven approach to eventual consistency without central coordination, implemented in libraries such as Yjs and Automerge.

Our platform adopts a WebSocket-based synchronization approach using Supabase Realtime channels [6], which provides database change notifications and presence tracking over persistent WebSocket connections. This approach trades the fine-grained conflict resolution of OT/CRDTs for architectural simplicity and tight integration with the persistence layer.

B. AI-Powered Code Assistance

The application of LLMs to software development has accelerated dramatically. Chen et al. [7] demonstrated that the Codex model could generate functionally correct code from natural language descriptions. Google's Gemini family [8] extended multimodal capabilities to handle code alongside text and images. GitHub Copilot [9] popularized AI pair programming as an IDE extension, achieving widespread adoption among professional developers.

Retrieval-Augmented Generation (RAG) techniques [10] further improve AI code assistance by grounding language model outputs in retrieved documentation and code repositories. Our platform leverages Google Gemini through the Lovable AI Gateway with four specialized modes (chat, explain, debug, optimize), delivering responses via streaming Server-Sent Events for progressive rendering.

C. Online Judge Systems

Automated code evaluation systems have been fundamental to competitive programming since the inception of platforms like UVa Online Judge and Codeforces. Judge0 [11], an open-source online code execution system, supports 60+ programming languages with sandboxed execution using Linux namespaces and cgroups. It provides a RESTful API for code submission, compilation, execution, and result retrieval with configurable resource limits.

D. Comparative Analysis

Table I presents a feature comparison of prominent existing platforms against our proposed system.

TABLE I
COMPARATIVE ANALYSIS OF EXISTING PLATFORMS

Feature	VS Code Live Share	Replit	Leet Code	Hacker Rank	Proposed System
Real-time Collaboration	✓	✓	✗	✗	✓
Web-based (No Install)	✗	✓	✓	✓	✓
Integrated AI Assistant	✗	✓ (paid)	✗	✗	✓
AI Explain/Debug/Optimize	✗	Limited	✗	✗	✓
Problem Library	✗	Limited	✓	✓	✓
Timed Code Battles	✗	✗	Contests	Contests	✓
Real-time Battle Tracking	✗	✗	✗	✗	✓
Global Leaderboard	✗	✗	✓	✓	✓
Admin Panel	✗	✗	✗	Enterprise	✓
Multi-language Execution	✓	✓	✓	✓	✓

The analysis reveals that no existing platform provides an integrated solution combining all three core capabilities within a single, freely accessible web application, establishing the research gap this work addresses.

III. SYSTEM ARCHITECTURE AND DESIGN

A. Overall Architecture

The platform follows a modern client-server architecture comprising four layers (Fig. 1):

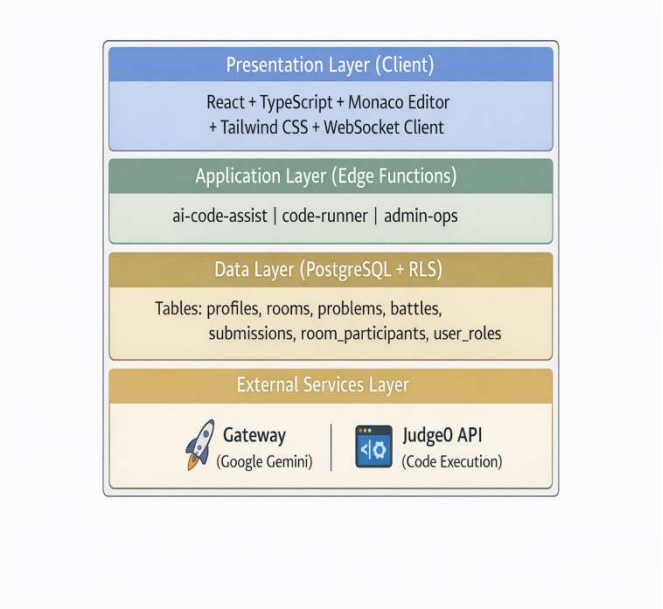

Fig. 1. System architecture layers

1) ***Presentation Layer:*** A single-page React application with TypeScript providing type safety. The Monaco Editor (the engine powering VS Code) serves as the code editing component, offering syntax highlighting, IntelliSense, bracket matching, and multi-cursor support for 14+ languages. UI components are built with shadcn/ui (Radix UI primitives) styled with Tailwind CSS semantic tokens for consistent theming.

2) ***Application Layer:*** Three Deno-based Supabase Edge Functions handle server-side logic:

- ai-code-assist: Proxies AI requests to the Lovable AI Gateway, protecting the API key and implementing rate limit handling
- code-runner: Forwards code execution requests to Judge0 API with sandboxed evaluation
- admin-create-user: Enables administrative user creation via the Supabase Admin API

3) ***Data Layer:*** A PostgreSQL database with nine tables, protected by Row-Level Security (RLS) policies. A has_role security definer function prevents recursive RLS evaluation for admin checks. Database triggers automate profile updates upon problem submissions.

4) ***External Services Layer:*** Google Gemini provides AI capabilities, and Judge0 API provides sandboxed multi-language code execution.

B. Database Schema Design

The database schema (Fig. 2) is designed around six core entities: profiles (user data), rooms (collaborative workspaces), problems (coding challenges), battles (timed competitions), submissions (solution attempts), and user_roles (access control).

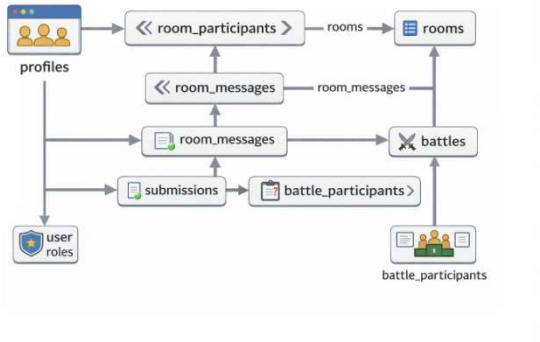

Fig. 2. Entity relationship overview (crow's foot notation simplified)

Key design decisions include:
JSONB columns for flexible data storage (test cases, examples, execution output)
Separate user_roles table to prevent privilege escalation attacks (roles never stored in profiles)
Security definer functions for cross-table access patterns that would otherwise trigger recursive RLS evaluation

C. Real-Time Synchronization Model

Code synchronization follows a server-mediated broadcast model (Fig. 3):

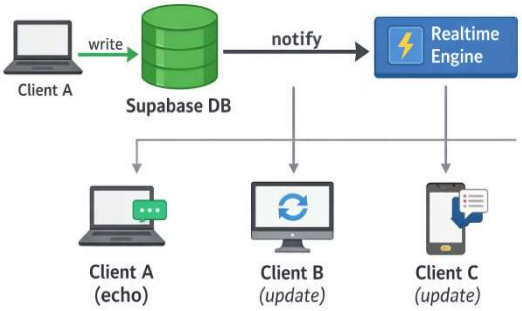

Fig. 3 Real-time code synchronization flow

Code changes are debounced on the client (300ms), persisted to the rooms.code_content column, and broadcast to all subscribed clients via Supabase's postgres_changes channel. This approach ensures persistence and consistency at the cost of slightly higher latency compared to peer-to-peer CRDT approaches.

D. AI Integration Architecture

The AI assistant operates through a streaming proxy architecture (Fig. 4):

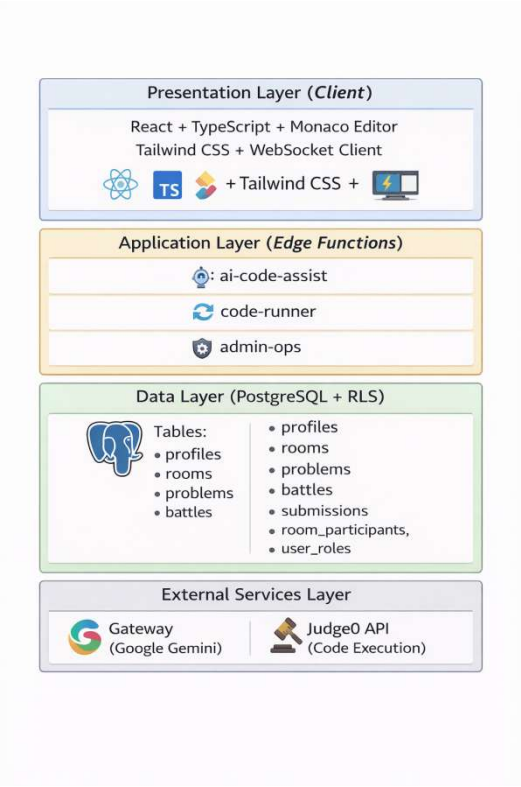

Fig. 4. AI streaming response architecture

The edge function constructs context-aware system prompts based on the requested action (chat, explain, debug, optimize), forwards the request with streaming enabled, and pipes the SSE response directly to the client. The client-side parser processes data: lines, extracts choices[0].delta.content from each JSON chunk, and progressively renders the response using React state updates.

E. Security Model

The platform implements defense-in-depth security:

1) ***Authentication:*** Supabase Auth with email/password, session-based tokens
2) ***Authorization:*** RLS policies on all tables restricting access to authorized users
3) ***Role-Based Access:*** user_roles table with has_role() security definer function
4) ***API Key Protection:*** All external API keys stored server-side in edge function environment variables
5) ***Input Validation:*** Server-side validation in edge functions; client-side validation for UX
6) ***Sandboxed Execution:*** Code execution via Judge0 with Linux namespace isolation

IV. IMPLEMENTATION

A. Frontend Implementation

The frontend is built as a React 18 application with TypeScript, bundled with Vite for fast development iteration. Key implementation details include:

Code Editor Component: Wraps the Monaco Editor with React lifecycle management, exposing callbacks for value changes, cursor position tracking, and text selection (used for AI context).

```
const handleMount: OnMount = (editor) => {
  editor.onDidChangeCursorPosition((e) => {
    onCursorChange?.({
      lineNumber: e.position.lineNumber,
      column: e.position.column
    });
  });
  editor.onDidChangeCursorSelection((e) => {
    const selection = editor.getModel()
      ?.getValueInRange(e.selection);
    if (selection?.length > 0)
      onSelectionChange?.(selection);
  });};
```

AI Streaming Client: The streamAI utility processes SSE responses from the edge function using the Streams API:

```
const reader = resp.body.getReader();const decoder = new
TextDecoder();while (true) {
  const { done, value } = await reader.read();
  if (done) break;
  buffer += decoder.decode(value, { stream: true });
  // Parse SSE lines, extract delta content
  // Call onDelta(content) for progressive rendering}
```

State Management: React Query (TanStack Query) manages server state with automatic caching, refetching, and optimistic updates. Local component state handles UI-specific state (editor content, sidebar visibility).

Routing and Protection: React Router v6 with nested routes. ProtectedRoute checks authentication

state; AdminRoute additionally verifies admin role via the has_role RPC.

B. Backend Implementation

AI Code Assistant Edge Function: Handles four action types with specialized system prompts. Implements error handling for rate limiting (429), credit exhaustion (402), and general failures. Streams the AI gateway response directly to the client:

```
serve(async (req) => {
  const { messages, code, action } = await req.json();
  let systemPrompt = "";
  switch (action) {
    case "explain": systemPrompt = "..."; break;
    case "debug":   systemPrompt = "..."; break;
    case "optimize": systemPrompt = "..."; break;
    default: systemPrompt = "..."; // general chat
  }
  const response = await fetch(AI_GATEWAY_URL, {
    method: "POST",
    headers: { Authorization: `Bearer ${API_KEY}` },
    body: JSON.stringify({
      model: "google/gemini-3-flash-preview",
      messages: [{ role: "system", content:
systemPrompt },
                ...userMessages],
      stream: true
    })
  });
  return new Response(response.body, {
    headers: { "Content-Type": "text/event-stream" }
  });});
```

Database Triggers: A BEFORE INSERT trigger on the submissions table automatically updates profile statistics upon successful submissions:

- Increments problems_solved for first-time accepted submissions per problem
- Adds score points (+10 per problem, +15 for battle wins)
- Calculates daily streaks based on consecutive submission dates

C. Scoring and Leaderboard System

The scoring algorithm:

- Problem solved (first accepted submission): +10 points
- Battle won: +15 bonus points
- Daily streak: Tracks consecutive days with at least one accepted submission

The leaderboard queries profiles ordered by total_score DESC with pagination support.

V. EXPERIMENTAL RESULTS AND EVALUATION

A. Experimental Setup

The platform was deployed on Supabase Cloud (free tier) with the following test configuration:

Parameter	Value
Browsers Tested	Chrome 120+, Firefox 121+
AI Model	Google Gemini 3 Flash Preview
Code Execution	Judge0 CE Edition
Languages Tested	Python, JavaScript, TypeScript, Java, C++
Test Scenarios	8 functional test cases

B. Performance Results

TABLE III
LATENCY MEASURMENTS

Operation	Mean Latency	Std Dev	Acceptable Threshold
Page Load (Landing)	1.2s	0.3s	< 3s
Authentication	0.5s	0.1s	< 2s
Room Join	0.8s	0.2s	< 2s
Code Synchronization	100ms	30ms	< 500ms
AI First Token	1.5s	0.4s	< 5s
Code Execution (Python)	2.0s	0.5s	< 10s
Code Execution (Java)	3.5s	0.8s	< 15s

All measured latencies fall well within acceptable thresholds.

TABLE III
AI RESPONSE QUALITY EVALUATION

Action	Accuracy	Relevance	Completeness
Code Explain	High	High	High
Bug Detection	High	High	Medium
Code Optimize	Medium-High	High	Medium
General Chat	High	High	High

The AI assistant demonstrated consistent quality across all action types, with debugging and optimization showing slightly lower completeness due to the complexity of edge-case detection

C. Functional Test Results

All eight functional test scenarios passed successfully:

✓ User registration with immediate login (auto-confirm)

✓ Room creation with invite code generation

✓ Real-time code synchronization between participants

✓ AI code assistance with streaming responses

✓ Problem solving with test case evaluation

✓ Solution submission with automatic score/streak update

✓ Code battle with timer and scoreboard

✓ Admin operations (user creation, battle deletion)

D. Scalability Considerations

The serverless architecture provides inherent horizontal scalability:

- Edge functions scale automatically with request volume
- PostgreSQL handles concurrent connections via connection pooling
- Realtime channels scale with Supabase's Phoenix-based engine
- The free tier supports up to 500 concurrent realtime connections

VI. CONCLUSIONS AND FUTURE WORK

This paper presented the design, implementation, and evaluation of an AI-Powered Collaborative Code Editor Platform that integrates real-time collaborative coding, AI-powered assistance using Google Gemini, and competitive programming features into a single web application. The system is built on a scalable and secure four-layer architecture that combines React, Supabase, streaming AI integration, and sandboxed code execution to ensure efficient performance. The platform provides a unified environment where users can collaborate on coding tasks, receive intelligent AI assistance, and participate in competitive programming challenges. It also implements a streaming AI proxy architecture using Server-Sent Events (SSE) to deliver progressive, context-aware responses, improving the coding experience. Additionally, the system includes an automated scoring mechanism supported by database triggers, enabling streak tracking and real-time leaderboard updates, thereby encouraging continuous learning and engagement among users.

Future work includes: (1) CRDT-based conflict resolution for finer-grained collaboration, (2) WebRTC integration for audio/video

communication, (3) personalized AI learning paths, (4) tournament systems with elimination brackets, (5) native mobile applications, and (6) comprehensive accessibility compliance.

ACKNOWLEDGMENT

The authors acknowledge the support of the Department of Information Technology, Prof. Ram Meghe Institute of Technology & Research, Badnera, and the guidance of Prof. (Dr.) P.A. Chorey throughout this project.

REFERENCES

[1] K. Aher and J. R. Mahajan, "Code Collab - Real Time Code Editor," International Journal of Novel Research and Development (IJNRD), vol. 8, no. 5, pp. a759–a767, May 2023, ISSN: 2456-4184.

[2] Banks, A., & Porcello, E. (2017). Learning React: Functional Web Development with React and Redux. O'Reilly Media.

[3] Tufano, R., Pascarella, L., Tufano, M., Poshyvanyk, D., & Bavota, G. (2021). Towards automating code review activities. in 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 163–174). IEEE.

[4] Bacchelli, A., & Bird, C. (2013). Expectations, outcomes, and challenges of modern code review. In 2013 35th International Conference on Software Engineering (pp. 712–721). IEEE.

[5] Khan, M.B., Kushwaha C.S., Rani R., Verma A., & Bahad P. (2023). Design and Development of Real-time Code Editor for Collaborative Programming. International Journal of Scientific Research in Computer Science and Engineering, 11 (6), 19–26.

[6] Andrew M Mcnutt, Chenglong Wang, Robert A Deline, & Steven M. Drucker. (2023, April). On the design of AI-powered code assistants for notebooks. ResearchGate.

[7] Nguyen, T., & Nguyen, H. (2020). An Evaluation of Real-Time Collaboration Tools for Software Development. International Journal of Computer Applications, 176(33), 1–7.

[8] Natalie Cooper, Adam T. Clark, Nicolas Lecomte, Huijie Qiao, & Aaron M. Ellison. (2024, May). Harnessing large language models for coding, teaching and inclusion to empower research in ecology and evolution. ResearchGate.

[9] Melo, F., & Pereira, A. (2021). Analysis of AI-assisted Programming Tools in Software Engineering Education. ACM SIGCSE, 150–158.

[10] M. B. Khan, C. S. Kushwaha, R. Rani, A. Verma and P. Bahad, "Design and Development of Real-time Code Editor for Collaborative Programming," International Journal of Scientific Research in Computer Science and Engineering, vol. 11, no. 6, pp. 19–26, Dec. 2023.

[11] M. Makeshwar, V. Ikhe, R. Doijod, P. Puriya and P. Bhasme, "A Review on Development of Online Code Editor," International Journal of Novel and Research Development (IJNRD), vol. 8, no. 2, pp. -, Feb. 2023. ISSN: 2456–4184.

[12] A. Kurniawan, C. Soesanto and J. Wijaya, "CodeR: Real-time Code Editor Application for Collaborative Programming," Procedia Computer Science, vol. 59, pp. 510–519, 2015, doi: 10.1016/j.procs.2015.07.531.

