IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155195

Secure Real-Time Coding Interview System with AI-Based Evaluation and Monitoring

Mr. J.R. Harshavardhan[1], Shreyas B[2], Yashas A[3], Yashwanth K[4], and Shiva Prakash J[5]

Associate Prof., Dept of CSE, K.S.School of Engineering & Management, Bengaluru, India[1]

Student, Dept of CSE, K.S.School of Engineering & Management, Bengaluru, India[2–5]

Abstract: The growing demand for remote technical hiring has accelerated the need for intelligent, unified platforms capable of conducting fair, scalable, and secure coding interviews. Traditional tools address only isolated aspects—collaboration, execution, or evaluation—leaving critical gaps in assessment integrity and real-time interactivity. This paper presents a Secure Real-Time Coding Interview System that integrates a conflict-free multi-user collaborative code editor, AI-based automated performance evaluation, Docker-sandboxed secure code execution, and behavioral monitoring into a single cloud-based platform. The system leverages WebSockets and CRDT/OT algorithms for real-time synchronization, Judge0 API for multi-language execution, and WebRTC/Socket.io for voice, video, and chat communication. Through a systematic review of five related works, six critical research gaps are identified, and a five-layer architecture is proposed to address them comprehensively. The platform requires no local installation and targets academic institutions and recruitment organizations seeking credible, interactive, and scalable online coding assessment infrastructure.

Keywords: Real-Time Collaboration; Coding Interview; AI Evaluation; WebSockets; CRDT; Docker; Monaco Editor; Secure Code Execution; Online Assessment; Socket.io; WebRTC; Node.js; React.js; Judge0 API; Behavioral Monitoring.

I. INTRODUCTION

The rapid growth of remote work and online hiring has transformed the technical interview landscape. Coding interviews—once conducted in person on whiteboards—now demand digital platforms that replicate and enhance the collaborative, evaluative, and proctoring capabilities of physical settings. Existing solutions are fragmented: standalone editors lack real-time collaboration; video platforms lack code execution; judge systems lack communication. None unify all these capabilities in a single coherent platform.

The emergence of WebSocket-based real-time communication, containerized code execution, and AI-driven assessment creates an opportunity to build an integrated coding interview system. However, achieving conflict-free multi-user editing, secure sandboxed execution, objective AI evaluation, and behavioral monitoring simultaneously within a single browser-based platform remains an open engineering challenge.

This paper proposes a Secure Real-Time Coding Interview System with AI-Based Evaluation and Monitoring. The system was motivated by direct observation of students engaged in group coding assignments (conducted March 26, 2026), where difficulty in real-time code sharing, concurrent-edit conflicts, and lack of integrated communication were consistently reported.

A. Objectives of This Work

- Critically review five recent works on collaborative code editors and online coding platforms.
- Identify research gaps in real-time collaboration, AI evaluation, secure execution, and proctoring.
- Propose an integrated five-layer system architecture addressing all identified gaps.
- Design a cloud-based, browser-accessible platform requiring zero local installation.
- Outline technology stack, methodology, and expected system advantages comprehensively.

II. PROBLEM IDENTIFICATION

Interaction with students during group coding assignments on March 26, 2026 revealed three recurring pain points. Students reported difficulty sharing code in real time across distributed team members, frustration when simultaneous edits by multiple users produced conflicting or overwritten code, and the absence of integrated communication tools forcing reliance on separate applications during collaborative sessions.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026
DOI: 10.17148/IJARCCE.2026.155195

These observations, corroborated by the literature reviewed in Section III, define the core problem: no existing platform unifies conflict-free real-time collaboration, AI-driven evaluation, secure sandboxed code execution, behavioral monitoring, and integrated communication in a single interview-grade system.

A. Core Problems Identified

- **No Effective Real-Time Collaboration:** Traditional editors do not support simultaneous multi-user editing, forcing teams to use version control systems that introduce delays and merge conflicts.
- **Communication Gap in Teamwork:** The absence of integrated chat and voice features makes it difficult for collaborators to discuss problems contextually while coding.
- **Difficulty Managing Concurrent Edits:** Simultaneous edits to the same code segment generate inconsistencies, degrading code quality and user experience.

III. RELEVANT LITERATURE

A. Paper [1]: CodeSync — Real-Time Collaborative Code Editor (Rudradev et al., 2026)

Rudradev et al. [1] proposed CodeSync, a browser-based collaborative code editor enabling multiple users to write, edit, and execute code simultaneously using WebSocket-based synchronization. The system supports multi-language execution and provides a shared workspace for group programming tasks.

The primary limitation is incomplete conflict resolution: concurrent edits by multiple users to the same code segment can produce inconsistent states. No version history, no AI-based evaluation, no behavioral monitoring, and no interview-specific features are present. The system targets collaborative learning rather than formal assessment.

B. Paper [2]: REALCODEAI — AI-Based Coding Interview Platform (Surendra et al., 2025)

Surendra et al. [2] proposed REALCODEAI, an AI-augmented real-time coding collaboration and interview platform integrating AI models for code evaluation, hint generation, and performance assessment. The platform targets recruitment and online assessment scenarios.

The primary limitation is computational overhead: reliance on large AI models introduces significant latency and resource consumption, making real-time responsiveness difficult to sustain at scale. The authors acknowledge that lightweight model alternatives are necessary for practical deployment.

C. Paper [3]: Collaborative Code Editor with Voice Chat (Anita et al., 2024)

Anita et al. [3] developed a real-time collaborative code editor augmented with integrated voice chat, enabling verbal communication during coding sessions. The addition of voice communication was shown to improve coordination and reduce misunderstandings in team programming tasks.

A significant limitation is the absence of conflict resolution: simultaneous edits lead to data overwrites. No AI-based evaluation, formal assessment workflow, proctoring, or permission-based access control for editing is implemented.

D. Paper [4]: Collaborative Code Editor for Coding Interviews (Reddy et al., 2024)

Reddy et al. [4] proposed a collaborative code editor specifically targeting the coding interview use case with features for candidate evaluation. The platform employs a mesh topology for peer-to-peer connectivity and supports multi-user interaction with live code state sharing.

Critical limitations arise from the mesh topology: bandwidth grows exponentially with participant count, causing severe scalability constraints. No AI-based evaluation, secure execution environment, or behavioral monitoring is integrated.

E. Paper [5]: CRDT-Based Collaborative Editor — Code Together (Santhi et al., 2024)

Santhi et al. [5] proposed Code Together, a web-based collaborative code editor implementing Conflict-free Replicated Data Types (CRDT) for concurrent multi-user editing. CRDTs guarantee eventual consistency without central coordination, enabling conflict-free simultaneous edits.

The primary limitation is implementation complexity: managing CRDT state across many concurrent users is computationally demanding and difficult to scale. No interview-specific, AI evaluation, or proctoring features are present.

IV. COMPARATIVE ANALYSIS OF EXISTING SYSTEMS

Table I presents a structured comparison of the five reviewed studies. Table II evaluates functional capabilities, revealing gaps addressed by the proposed system.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155195

TABLE I. COMPARISON OF SURVEYED RESEARCH

Ref.	Research Focus	Technique(s)	Key Limitations
[1] Rudradev 2026	CodeSync real-time collaborative editor	WebSockets, Node.js	Incomplete conflict resolution; no AI; no monitoring
[2] Surendra 2025	AI-augmented coding interview platform (REALCODEAI)	Large AI models, WebSockets	High latency; resource-intensive; not scalable
[3] Anita 2024	Collaborative editor with voice chat	WebSockets, WebRTC	No conflict resolution; no AI; no proctoring
[4] Reddy 2024	Interview-oriented collaborative editor, mesh P2P	Mesh topology, WebSockets	Bandwidth $O(n^2)$; no AI; no sandboxed exec
[5] Santhi 2024	CRDT-based collaborative editor (Code Together)	CRDT, WebSockets	High CRDT complexity; no interview features; no AI

TABLE II. FEATURE AND CAPABILITY COMPARISON

Paper	RT Collab	Conflict Free	AI Eval	Secure Exec	Proctoring	Voice Video	Interv. Grade
[1] Rudradev 2026	Yes	No	No	No	No	No	No
[2] Surendra 2025	Yes	Partial	Yes	Partial	No	No	Partial
[3] Anita 2024	Yes	No	No	No	No	Yes	No
[4] Reddy 2024	Yes	No	No	No	No	No	No
[5] Santhi 2024	Yes	Yes (CRDT)	No	No	No	No	No
Proposed System	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**

V. GAP ANALYSIS

Systematic review and comparative analysis of the five surveyed papers reveals six critical research gaps motivating the proposed system:

A. Absence of AI-Based Automated Evaluation

None of the surveyed papers [1][3][4][5] incorporate AI-driven evaluation of code quality, correctness, or efficiency. Surendra et al. [2] include AI evaluation but acknowledge unacceptable latency at scale. No existing work delivers lightweight, real-time AI assessment integrated within a collaborative interview platform.

B. Lack of Conflict-Free Concurrent Editing

Only Santhi et al. [5] implement CRDT-based conflict resolution, but at the cost of implementation complexity that limits scalability. Papers [1][3][4] suffer data overwrites on concurrent edits. No work successfully combines conflict-free editing with interview features and AI evaluation.

C. Absence of Secure Sandboxed Code Execution

None of the five surveyed papers implement a secure, sandboxed execution environment. Code execution, where present, relies on browser-based or unverified managed services. Docker-based isolation—essential for preventing malicious code from affecting host systems in an adversarial interview context—is entirely absent from existing literature.

D. No Behavioral Monitoring or Proctoring

None of the surveyed papers implement any form of behavioral monitoring, proctoring, or anti-cheat mechanisms. In a formal interview or assessment context, monitoring of candidate activity (tab switching, clipboard use, keystroke dynamics) is essential for maintaining assessment integrity—a gap unaddressed across all five works.

DOI: 10.17148/IJARCCE.2026.155195

E. Scalability Limitations

Reddy et al. [4] explicitly identify scalability as a critical failure mode of their mesh topology. Surendra et al. [2] face resource scalability constraints from large AI models. Santhi et al. [5] acknowledge CRDT state-management complexity at scale. No existing paper delivers a provably scalable unified interview platform.

F. Absence of Interview-Grade Integrated Communication

While Anita et al. [3] integrate voice chat, no existing work provides a complete suite of interview-grade communication tools—combining real-time text chat, voice, and video conferencing within the same platform alongside code collaboration, execution, and AI evaluation.

VI. PROPOSED SYSTEM DESIGN

A. System Overview

The proposed Secure Real-Time Coding Interview System addresses all six identified gaps through an integrated cloud-based platform that: (i) enables conflict-free real-time multi-user code editing; (ii) executes user-submitted code securely within Docker containers; (iii) evaluates candidate performance through an AI-driven assessment engine; (iv) monitors behavioral signals for proctoring; and (v) facilitates in-context communication via integrated chat, voice, and video.

B. Five-Layer Architecture

The system follows a five-layer architecture:

- **Presentation Layer:** React.js/Next.js frontend with Monaco Editor; browser-accessible with zero local installation.
- **Collaboration Layer:** Socket.io WebSockets with CRDT/OT algorithms ensuring conflict-free real-time synchronization across all clients.
- **Execution Layer:** Docker-containerized sandboxed code execution with Judge0 API supporting 60+ programming languages, with time and memory limits enforced.
- **AI Evaluation Layer:** Lightweight AI microservice assessing code correctness, complexity, style, and edge-case handling; generates structured evaluation reports.
- **Communication & Monitoring Layer:** Socket.io chat, WebRTC voice/video conferencing, and behavioral proctoring module tracking tab switches, clipboard events, and keystroke patterns.

C. Conflict-Free Collaborative Editing

Concurrent edits by multiple users are managed through CRDT (Conflict-free Replicated Data Types) combined with Operational Transformation (OT) fallback. CRDT ensures eventual consistency without central coordination by representing the document as a partially-ordered set of operations that can be merged deterministically. This directly addresses the overwrite failures of [1][3][4] and the complexity limitations of [5].

D. Key Platform Capabilities

- **Editing:** Multi-user CRDT/OT sync, Monaco Editor, syntax highlighting, IntelliSense, 60+ language support.
- **Execution:** Docker sandboxing, Judge0 API, time/memory limits, real-time output streaming, multi-language execution.
- **AI Evaluation:** Correctness scoring, time/space complexity analysis, code style linting, edge-case coverage, structured report generation.
- **Proctoring:** Tab-switch detection, clipboard event monitoring, keystroke dynamics, webcam activity tracking, session audit logs.
- **Communication:** Socket.io text chat, WebRTC peer-to-peer voice and video, in-context annotation and code commenting tools.

VII. METHODOLOGY

A. System Development Approach

The system is developed using an iterative, component-driven methodology. Each of the five architectural layers is designed, implemented, and tested independently before integration. The frontend employs React.js component architecture; the backend follows RESTful API design with Express.js; and real-time functionality is built on Socket.io event-driven programming.

B. Real-Time Collaboration Implementation

WebSocket connections via Socket.io maintain persistent bidirectional communication between all connected clients. CRDT document representation encodes each character insertion and deletion as an operation with a unique identifier

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026
DOI: 10.17148/IJARCCE.2026.155195

and causal history. On receiving a remote operation, the local CRDT state is merged deterministically, eliminating the need for a central conflict-resolution server and enabling offline-first edit merging consistent with [5].

C. Secure Code Execution Pipeline

User-submitted code is packaged with language-specific metadata and dispatched to a Docker execution worker. The worker spawns an isolated container with enforced CPU time limits (default: 5 seconds), memory limits (default: 256 MB), and no network access. Standard output, error output, and exit codes are captured and returned. Alternatively, the Judge0 API provides a managed multi-language execution service with equivalent isolation guarantees.

D. AI Evaluation Module

The AI evaluation microservice accepts submitted code and a problem specification as input. It runs the code against a suite of hidden test cases, measures time and space complexity via instrumented execution, applies static analysis for style and correctness, and generates a structured JSON evaluation report. Lightweight transformer-based models handle natural language feedback generation, directly addressing the latency problem of [2] by constraining model size for real-time operation.

E. Behavioral Monitoring

A JavaScript browser agent installed in the candidate's session captures: tab/window focus events, clipboard read/write operations, paste event frequency, mouse movement entropy, and keystroke timing patterns. These signals are aggregated into a per-session integrity score and surfaced on the interviewer's monitoring dashboard in real time.

VIII. TECHNOLOGY AND TOOLS USED

A. Software Requirements

- **Frontend:** HTML5, CSS3, JavaScript (ES6+), React.js / Next.js, Monaco Editor
- **Backend:** Node.js, Express.js, REST API architecture
- **Real-Time:** Socket.io (WebSockets), WebRTC peer-to-peer
- **Database:** Firebase (real-time sync) / PostgreSQL (structured data)
- **Code Execution:** Docker (sandboxed isolation) / Judge0 API (managed)
- **AI Evaluation:** Python microservice, lightweight transformer models, REST API
- **Deployment:** AWS / Vercel (cloud-hosted, zero local install required)

B. Hardware Requirements

- **Processor:** Intel Core i5 (8th Gen or above) / equivalent AMD
- **RAM:** Minimum 8 GB (16 GB recommended for Docker-based development)
- **Storage:** Minimum 256 GB SSD
- **Network:** Stable broadband internet connection (minimum 10 Mbps)
- **Platform:** Laptop or Desktop system; end users require only a modern browser

IX. EXPECTED OUTCOMES AND BENEFITS

A. Collaborative Accuracy

The CRDT/OT-based conflict resolution mechanism is expected to eliminate overwrite conflicts in concurrent editing scenarios, delivering consistent document state across all participants irrespective of network latency.

B. Evaluation Performance

The lightweight AI evaluation module is designed to deliver per-submission assessment results within 2–5 seconds, addressing the scalability and latency limitations identified in Surendra et al. [2]. Evaluation metrics include correctness, complexity, style quality, and edge-case coverage.

C. Security and Integrity

Docker-containerized execution enforces strict resource limits and network isolation, preventing malicious code from affecting host systems. Behavioral proctoring signals provide interviewers with objective integrity indicators, reducing reliance on honor-based assessment.

D. Accessibility and Scalability

As a cloud-deployed, browser-based platform requiring zero local installation, the system is accessible to academic institutions and recruitment organizations without infrastructure investment. The Socket.io hub-and-spoke topology replaces the $O(n^2)$ mesh of [4], enabling linear scalability with participant count.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026
DOI: 10.17148/IJARCCE.2026.155195

X. CONCLUSION AND FUTURE WORK

This paper presented the design, objectives, and methodology of a Secure Real-Time Coding Interview System with AI-Based Evaluation and Monitoring. Through systematic review and comparative analysis of five related works, six critical research gaps were identified: absence of AI-based automated evaluation, lack of conflict-free concurrent editing at scale, no secure sandboxed execution environment, no behavioral monitoring or proctoring, scalability limitations, and fragmented interview-grade communication.

The proposed system addresses all six gaps through a five-layer architecture combining React.js/Node.js, CRDT/OT-based conflict-free collaboration via Socket.io, Docker/Judge0 secure code execution, a lightweight AI evaluation microservice, WebRTC communication, and a behavioral proctoring module. The platform is cloud-deployed, browser-accessible, and requires no local installation.

Future work will focus on empirical validation of the AI evaluation module through user studies measuring assessment consistency and candidate experience, extension to group interview formats, and integration of reinforcement learning for adaptive question selection based on real-time candidate performance signals.

XI. ACKNOWLEDGEMENT

The authors would like to extend their sincere gratitude to the Department of Computer Science and Engineering, K. S. School of Engineering and Management (KSSEM), Bengaluru, for providing an excellent research environment and necessary institutional support for this work. We thank our project guide and faculty members for their invaluable guidance and encouragement throughout the preparation of this paper.

We would also like to acknowledge the contributions of the open-source research community and the authors of the reviewed studies whose work forms the foundation of this survey.

REFERENCES

[1]. Myadara Rudradev, Bagula Sai Kishore, and Sabnekar Asha Jyothi, “Real-Time Collaborative Code Editor,” International Journal of Creative Research Thoughts (IJCRT), Vol. 14, Issue 1, Jan. 2026, ISSN: 2320-2882.
[2]. Hiware Meher Surendra, Harkal Swapnil Ratan, Ghallal Goraksh Bajirao, and Khatal Kavita B., “Real Time Code Collaboration and Interview Platform,” IJSRSET, Vol. 12, Issue 6, Nov.–Dec. 2025, pp. 150–156.
[3]. P. Anita, Sakithyan A., Dilip S., Nikhil R., Rithish S., and Maneesh Aravind S., “Real Time Collaborative Code Editor,” IJSREM, Vol. 08, Issue 07, July 2024, ISSN: 2582-3930.
[4]. N. V. Raja Sekhar Reddy, Satvik Pothukuchi, and Rahul Sivapuram, “Real Time Collaborative Code Editor,” IJRASET, Vol. 12, Issue VIII, Aug. 2024.
[5]. N. Jaya Santhi, D. Sireesha, E. Vindhya, and D. Naga Jyothi, “Collaborative Code Editor Using Web Application,” IARJSET, Vol. 11, Issue 3, Mar. 2024.
[6]. G. Ellis et al., “Operational Transformation in Real-Time Group Editors,” in Proc. ACM CSCW, 1998, pp. 59–68.
[7]. M. Shapiro et al., “Conflict-free Replicated Data Types,” in Proc. SSS 2011, LNCS vol. 6976, Springer, 2011, pp. 386–400.
[8]. Docker Inc., “Docker: Empowering App Development for Developers,” [Online]. Available: https://www.docker.com. [Accessed: May 2026].
[9]. Judge0 LLC, “Judge0 — Open-Source Online Code Execution System,” [Online]. Available: https://judge0.com. [Accessed: May 2026].
[10]. Microsoft Corp., “Monaco Editor,” [Online]. Available: https://microsoft.github.io/monaco-editor. [Accessed: May 2026].

