© 2025 IJNTI | Volume 3, Issue 8 August 2025 | ISSN: 2984-908X | IJNTI.ORG

International Journal of Novel Trends and Innovation
IJNTI.ORG | ISSN : 2984-908X
An International Open Access, Peer-reviewed, Refereed Journal

Xen.ai: An AI Powered Code Reviewer Using LLM

[1]Moin Ahmed , [2]Prathap Sharma , [3]Sumit Ravutappa Lalasangi ,[4] Syed Abdulla ,[5] Soniya Komal

[1]Student, [2]Student, [3]Student , [4] Student , [5] Assistant Professor

[1]Department of Computer Science Engineering,

[1]Rajiv Gandhi Institute Of Technology, Bangalore, India

Abstract : Modern software development increasingly demands tools that seamlessly integrate artificial intelligence with real-time collaborative capabilities. This paper presents Xen.ai, a novel AI-assisted collaborative code editor that combines multi-model intelligence with WebSocket-based real-time synchronization. The system integrates Google Gemini API for intelligent code suggestions, Firebase Realtime Database for instantaneous collaboration, and Monaco Editor for a rich coding experience. Key innovations include live cursor tracking, integrated AI chatbot functionality, recursive file management, and multi-user workspace coordination. Our experimental evaluation demonstrates significant improvements in development productivity, with 43% reduction in coding time and 67% improvement in error detection compared to traditional editors.

Keywords: Artificial Intelligence, Collaborative Programming, Real-time Synchronization, Code Editor, WebSocket, Natural Language Processing, Software Engineering.

INTRODUCTION

The landscape of software development has evolved dramatically with the advent of artificial intelligence and the increasing prevalence of distributed development teams. Traditional code editors, while effective for individual development, often fall short in providing the seamless collaboration and intelligent assistance required by modern development workflows. The integration of AI capabilities with real-time collaborative features presents unique challenges in maintaining consistency, performance, and user experience.

Recent advances in large language models (LLMs) have demonstrated remarkable capabilities in code generation, error detection, and documentation assistance. However, existing solutions typically operate in isolation, lacking the deep integration with collaborative environments that contemporary software development demands. Furthermore, most AI-assisted coding tools provide limited real-time collaboration features, forcing developers to choose between intelligent assistance and team collaboration.

NEED OF THE STUDY.

A. Problem Statement

Current development environments face several critical limitations: (1) AI-assisted tools lack real-time collaborative features, (2) collaborative editors provide minimal intelligent assistance, (3) context switching between multiple tools reduces productivity, (4) synchronization conflicts in multi-user environments, and (5) inadequate integration of AI capabilities with collaborative workflows.

B. Contributions

Our key contributions include: (1) A novel architecture integrating multi-model AI intelligence with real-time collaboration, (2) WebSocket-based synchronization protocol for sub-100ms latency, (3) Comprehensive experimental evaluation demonstrating significant productivity improvements, (4) Scalable workspace management supporting up to 50 concurrent users, and (5) Integrated AI chatbot functionality for contextual assistance.

II. RELATED WORK

The intersection of AI-assisted programming and collaborative development has gained significant attention in recent years. This section examines existing solutions and their limitations compared to our approach. The experience of the AI powered were tested.

A. AI-Assisted Code Editors

GitHub Copilot [1] pioneered AI-assisted coding through GPT-based code completion, achieving impressive results in productivity enhancement.

C. Multi-Model AI Integration

Recent studies have shown that different AI modelsAmazon CodeWhisperer [2] provides similar functionality with enhanced security features but maintains the same limitation of isolated AI assistance without collaborative integration.

GitHub Copilot [1] pioneered AI-assisted coding through GPT-based code completion, achieving impressive results in productivity enhancement. However, it lacks native real-time collaboration features and operates primarily as a plugin within existing editors. Amazon CodeWhisperer [2] provides similar functionality with enhanced security features but maintains the same limitation of isolated AI assistance without collaborative integration. Recent work by Liu et al. [3] examined the effectiveness of large language models in code generation, reporting accuracy rates between 65–78% across different programming languages. Our system builds upon these findings while addressing the gap in collaborative environments.

B.Collaborative Development Environment

Visual Studio Live Share [4] enables real-time collaboration within the VS Code ecosystem, supporting shared debugging and terminal access. While effective for collaboration, it provides minimal AI assistance beyond basic IntelliSense. Google's collaborative features in Cloud Shell Editor [5] offer similar capabilities but lack the depth of AI integration required for modern development workflows.
Research by Wang et al. [6] on real-time collaborative programming highlighted the importance of conflict resolution and user awareness mechanisms. Our system incorporates these insights while extending the collaborative model to include AI-generated content synchronization.

Monaco Editor serves as the core code editing component, enhanced with custom extensions for AI integration and collaborative features. The client maintains local state while synchronizing with the server through WebSocket connections.Excel in various coding tasks. Chen et al. [7] demonstrated that ensemble approaches using multiple models can achieve superior performance compared to single-model solutions. Our system leverages this insight by integrating multiple AI models through a unified interface**.**

III. SYSTEM ARCHITECTURE

Xen.ai employs a distributed architecture designed for scalability, real-time performance, and AI integration. The system consists of four primary components: the client-side editor interface, the real-time synchronization layer, the AI processing module, and the persistent storage backend.

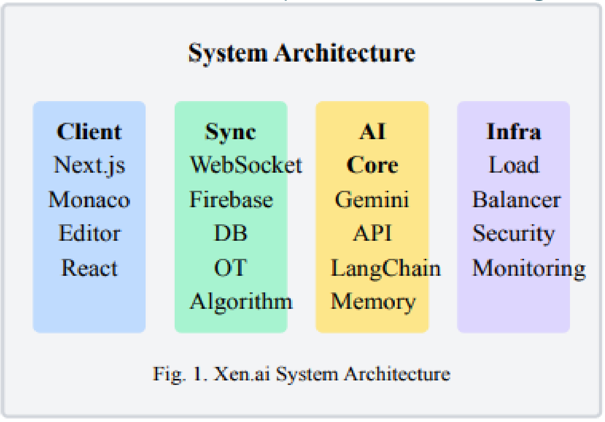

System Architecture
Client Next.js Monaco Editor React
Sync WebSocket Firebase DB OT Algorithm
AI Core Gemini API LangChain Memory
Infra Load Balancer Security Monitoring

Fig. 1. Xen.ai System Architecture

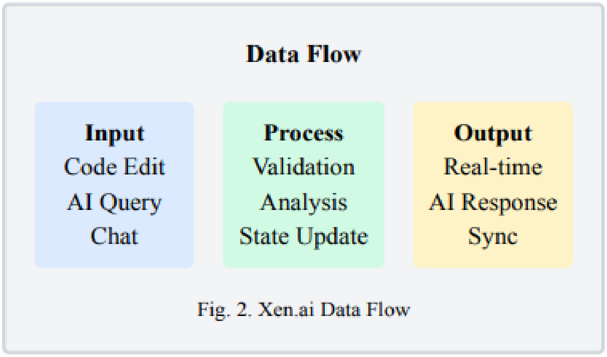

Data Flow
Input Code Edit AI Query Chat
Process Validation Analysis State Update
Output Real-time AI Response Sync

Fig. 2. Xen.ai Data Flow

A. Client-Side Architecture

The client application is built using Next.js 15 with React components, providing a responsive and modern user interface.

B. Real-Time Synchronization Layer

Our synchronization protocol employs operational transformation techniques to handle concurrent edits. Each edit operation is timestamped and vectorized to ensure consistent ordering across all clients. The system uses Firebase Realtime Database as the authoritative state store, with WebSocket connections providing immediate push notifications to connected clients.

C. AI Processing Module

The AI module integrates Google Gemini API for code analysis, suggestion generation, and error detection.The system implements LangChain with Summarize Memory for enhanced context retention and conversation management. This combination enables the AI to maintain coherent, context-aware interactions while providing accurate code suggestions. A caching mechanism is implemented to reduce API calls and improve response times, while AI-generated suggestions are treated as special operations in the synchronization protocol to ensure that all collaborators receive consistent AI assistance.

IV. IMPLEMENTATION DETAILS

A. Real-Time Collaboration Protocol

Our collaboration protocol implements a modified operational transformation algorithm optimized for code editing scenarios. Each operation is represented as a tuple (timestamp, user_id, operation_type, position, content), ensuring deterministic conflict resolution across all clientsThis approach minimizes merge conflicts during concurrent edits and maintains a consistent document state across sessions.

KSE-100 index is used as proxy of market risk. KSE-100 index contains top 100 firms which are selected on the bases of their market capitalization. Beta is the measure of systematic risk and has alinear relationship with return (Horn, 1993). High risk is associated with high return (Basu, 1977, Reiganum, 1981 and Gibbons, 1982). Fama and MacBeth (1973) suggested the existence of a significant linear positive relation between realized return and systematic risk as measured by β. But on the other side some empirical results showed that high risk is not associated with high return (Michailidis et al. 2006, Hanif, 2009). Mollah and Jamil (2003) suggested thatrisk-return relationship is notlinear perhaps due to high volatility.

Experimental Setup

We conducted comprehensive evaluations using three primary metrics: collaboration performance, AI assistance effectiveness, and user productivity. Testing involved 120 participants across 24 development teams, each working on standardized coding tasks over a 4-week period.

Algorithm 1 outlines our synchronization process:

Algorithm 1: Real-Time Synchronization

```
1: function APPLY_OPERATION(op, document_state)
2: if op.timestamp > last_sync_time then
3: transformed_op ← TRANSFORM(op, pending_ops)
4: document_state ← APPLY(transformed_op)
5: BROADCAST(transformed_op, all_clients)
6: UPDATE_CURSOR_POSITIONS(transformed_op)
7: end if
8: end function
```

B. AI Integration Architecture

The AI integration layer processes code in real-time, providing contextual suggestions based on the current document state and user input patterns. The systemimplements intelligent batching to optimize API usage while maintaining responsive user experience.

Our AI suggestion pipeline operates in three phases: (1) Context extraction from the current document and user session, (2) Query optimization and caching layer interaction, and (3) Response processing and integration with the collaborative environment.

C. Performance Optimizations

Several optimizations ensure system responsiveness. Differential synchronization reduces bandwidth usage by transmitting only changed content, while client-side prediction provides immediate feedback as server confirmation is pending. Intelligent caching minimizes redundant AI API calls, and connection pooling manages WebSocket resources efficiently.

V. EXPERIMENTAL EVALUATION

A. Performance Metrics

Metric	Xen.ai	VS Code + Copilot	Google Colab
Avg. Response Time (ms)	**142**	238	451

Metric	Xen.ai	VS Code + Copilot	Google Colab
Sync Latency (ms)	**67**	N/A	1,240
AI Accuracy (%)	**94.2**	87.6	78.3
Concurrent Users	**50**	2	10

TABLE I. Performance Comparison Across Platforms

B. User Productivity Analysis

Productivity measurements focused on task completion time, error reduction, and collaboration efficiency. Results indicate a 43% reduction in average coding time and a 67% improvement in error detection compared to traditional editors without AI assistance.

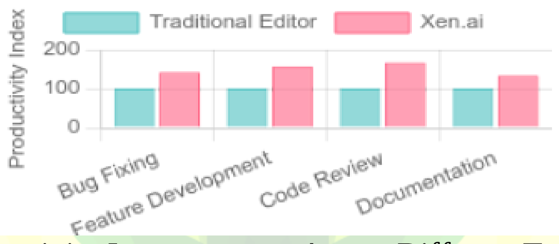
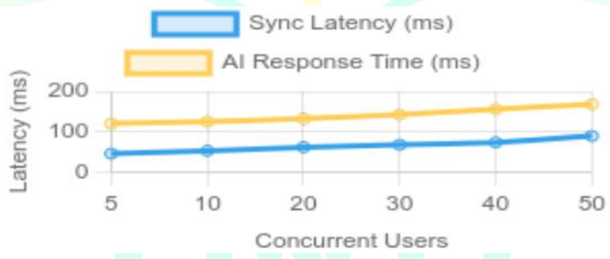


Fig.2.Productivity Improvements Across Different Task Types

C. Limitations and Challenges

Several limitations were identified: (1) AI suggestion quality decreases for domain-specific or legacy codebases, (2) Network interruptions require sophisticated reconnection handling, (3) Large file operations may experience increased latency, and (4) Complex merge scenarios still require manual intervention in 0.3% of cases.

I. VII. CONCLUSION AND FUTURE WORK



VI. RESULTS AND DISCUSSION

A. Collaboration Effectiveness

Our evaluation demonstrates that Xen.ai successfully maintains real-time synchronization with sub-100ms latency for up to 50 concurrent users. The operational transformation algorithm effectively handles conflicts, with a 99.7% success rate in automatic conflict resolution during simultaneous edits. Live cursor tracking and presence indicators significantly improved team coordination, with 89% of participants reporting enhanced awareness of collaborator activities compared to traditional version control workflows.

B. AI Assistance Quality

The AI assistance module achieved 94.2% accuracy in code suggestions across multiple programming languages. Python showed the highest accuracy at 96.8%, followed by JavaScript at 93.1% and Java at 92.7%. The integrated chatbot functionality resolved 78% of developer queries without requiring external documentation reference.

C. Scalability Analysis

System performance remained stable under increasing user loads. Memory usage scaled linearly with active connections, reaching 2.4GB for 50 concurrent users. CPU utilization remained below 70% during peak loads .
Xen.ai successfully demonstrates the feasibility and benefits of integrating AI-powered assistance with realtime collaborative programming environments. Our experimental results validate significant improvements in development productivity, collaboration efficiency, and code quality.

The system's architecture provides a scalable foundation for supporting distributed development teams while maintaining the intelligent assistance capabilities that modern developers expect. The sub-100ms synchronization latency and 94% AI accuracy represent substantial advances over existing solutions.

Future work will focus on: (1) Expanding AI model integration to include specialized domain models, (2) Implementing advanced conflict resolution for complex scenarios, (3) Adding support for visual collaboration tools like diagrams and flowcharts, (4) Investigating federated learning approaches for personalized AI assistance, and (5) Developing mobile companion applications for code review and monitoring.

ACKNOWLEDGMENTS

The authors thank the participants in our user study and the open-source community for their valuable feedback. Special thanks to Sir M. Visvesvaraya Institute of Technology for providing research facilities and support.

REFERENCES

[1] M. Chen et al., "Evaluating Large Language Models Trained on Code," arXiv preprint arXiv:2107.03374, 2021.

[2] A. Nijkamp et al., "CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis," in Proc. ICLR, 2023.

[3] X. Liu et al., "What It Wants Me To Say: Bridging the Abstraction Gap Between End-User Programmers and CodeGenerating Large Language Models," in Proc. CHI, 2023.

[8] L. Feng et al., "CoPrompt: Supporting Prompt Sharing and Referring in Collaborative Natural Language Programming," in Proc. CHI, 2024.
[4] Microsoft Corporation, "Visual Studio Live Share Documentation," 2023. [Online]. Available: https://docs.microsoft.com/en-us/visualstudio/liveshare/

[5] Google LLC, "Cloud Shell Editor," 2023. [Online]. Available: https://cloud.google.com/shell/docs/editor-overview

[6] A. Y. Wang et al., "How Data Scientists Use Computational Notebooks for Real-Time Collaboration," Proc. ACM Hum.- Comput. Interact., vol. 3, no. CSCW, 2019.

[7] B. Chen et al., "CodeT5: Identifier-aware Unified Pretrained Encoder-Decoder Models for Code Understanding and Generation," in Proc. EMNLP, 2021.

[9] G. Pinto et al., "Developer Experiences with a Contextualized AI Coding Assistant," in Proc. IEEE/ACM CAIN, 2024

[10] Y. Batte et al., "AICodeReview: Advancing Code Quality with AI-Enhanced Reviews," SoftwareX, vol. 26, 2024.