Grenze International Journal of Engineering and Technology, January Issue

Collaborative Programming Platform with Integrated AI Debugger and Code Recommender

Suvalakshmi K[1], Bizu B[2], Roopa C[3], Nikilhari R[4], Sandhish J[5] and Mohamed Maahij N[6]
[1-6]Kongu Engineering College/Department of CSE, Perundurai, Tamil Nadu, India
Email: suva1798@gmail.com, bizucse@gmail.com, roopavivin@gmail.com, nikilharir.22cse@kongu.edu, sandhishj.22cse@kongu.edu, mohamedmaahijn.22cse@kongu.edu

***Abstract*— Maintaining efficiency and accuracy is a critical challenge in collaborative software development. This paper presents a hybrid programming platform that integrates real-time collaboration with AI-driven debugging and code recommendation. The system combines CodeT5 for rapid syntax correction with PhiCoder-2 for semantic analysis, enabling both speed and contextual accuracy. A WebSocket-based shared workspace supports multi-user editing, version control, and conflict resolution, while the AI (Artificial Intelligence) modules provide instant feedback to reduce debugging effort. Experimental results on the CodeXGLUE dataset show a 25% improvement in error detection accuracy and a 30% reduction in debugging time compared to baseline systems. Comparative analysis demonstrates that the proposed platform uniquely unifies WebSocket-enabled collaboration and intelligent debugging in a single environment. This work contributes a practical framework for enhancing developer productivity and lays the groundwork for scalable, AI-assisted programming environments.**

***Index Terms*— Collaborative Programming, Real-Time Code Editor, AI Debugging, Transformer Models, WebSocket Synchronization, Software Development Productivity.**

I. INTRODUCTION

The rise of collaborative development environments has transformed modern software engineering practices. While traditional IDEs (Integrated Development Environment) such as Eclipse and Visual Studio provide powerful standalone development capabilities, they remain limited in supporting distributed teams, real-time collaboration, and AI-assisted coding [1]. The increasing reliance on remote teamwork, coupled with the complexity of modern software systems, highlights the need for intelligent, collaborative solutions.

Artificial Intelligence (AI) has emerged as a key enabler in software development, providing intelligent tools for code completion, debugging, and error detection. Platforms like GitHub Copilot demonstrate the potential of AI assistance but lack integrated collaboration features [2]. Similarly, collaborative tools like Google Colab and Replit support teamwork but provide limited debugging intelligence [1]. This gap motivates the development of a unified platform that combines AI-powered assistance with real-time collaborative coding. This paper introduces a novel collaborative programming platform that integrates AI models for syntax correction, logical debugging, and intelligent code recommendations [3]. By merging real-time collaboration with AI-enhanced workflows, the platform supports developers in writing cleaner, error-free code while improving teamwork efficiency.

Grenze ID: 01.GIJET.12.1.235

The system architecture incorporates WebSocket-based communication for real-time synchronization, transformer-based AI models for intelligent assistance, and a modular design that supports multiple programming languages.

II. Related Work

Artificial Intelligence has significantly transformed software engineering by enabling intelligent support in code review, debugging, and autocompletion. Feng et al. introduced CodeBERT, a pre-trained model for programming and natural languages, which served as a foundation for several advanced code intelligence systems [4]. Building on this foundation, Zhang et al. developed AICodeReview: Advancing Code Quality with AI-enhanced Reviews combined CodeBERT with Gemini in a web-based framework, achieving 87% accuracy in syntax error detection, 82% in logical errors, and 78% in security vulnerabilities [5]. These results demonstrated the ability of AI models to reduce manual review effort while ensuring reliable code quality.

The evolution of code completion tools has also been noteworthy. Chen et al. evaluated large-scale models such as Codex, showing that deep learning-based approaches could achieve over 90% accuracy in code suggestion while reducing typing effort by nearly 45% [2]. Similarly, Svyatkovskiy et al. IntelliCode Compose leveraged transformer models to provide intelligent autocompletion, streamlining the developer workflow and reducing error rates [6]. Beyond accuracy, context-aware debugging has become a focus area. Vaswani et al. established the foundational transformer architecture with logical reasoning capabilities, enabling detection of deeper semantic issues and providing more reliable fixes for complex code [7]. Recent work by Shao et al. such as RepoDebug highlighted the importance of multi-language datasets for evaluating debugging systems, showing that AI performance varies across error types and programming languages [8].

Other approaches such as AutoSD introduced explainable debugging methods by leveraging scientific reasoning with LLMs [9], while Kumar et al. applied secure AI-driven review systems on GitHub pull requests with Bugdar [10]. Ahmad et al. proposed unified pre-training strategies for program understanding to improve model adaptability across tasks [11]. Meanwhile, Liu et al. evaluated large language models for code review, further demonstrating the need for robust AI systems in collaborative development [12].

Overall, the literature establishes that while syntax-level debugging and code completion are highly effective with models like CodeT5 [3] and Codex [2], greater emphasis on semantic reasoning and real-world validation is essential. These studies provide the foundation for developing collaborative platforms that integrate AI-driven debugging and recommendation features, making programming more efficient, accurate, and accessible.

III. Motivation

Contemporary software development faces three critical challenges that motivate this research:

- Inefficient Debugging Processes: Manual error detection and correction consume significant development time, with developers spending approximately 50% of their time debugging rather than creating new functionality.
- Limited Collaborative Tools: Existing collaborative programming platforms lack sophisticated debugging capabilities, forcing teams to switch between multiple tools for code editing, collaboration, and debugging.
- Fragmented AI Integration: AI-powered programming assistance exists primarily as isolated plugins or separate tools, preventing seamless integration with collaborative workflows.

The convergence of these challenges creates substantial inefficiencies in team-based software development. This research is motivated by the potential to significantly improve developer productivity through intelligent, collaborative programming environments that provide real-time AI assistance within a unified platform.

IV. Problem Domain

The problem domain encompasses the integration of collaborative programming environments with AI-assisted debugging and recommendation systems. Current solutions address these aspects in isolation: collaborative platforms excel in multi-user interaction but lack intelligent assistance, while AI debugging tools provide sophisticated analysis but operate independently of collaborative workflows.

The domain complexity involves several technical challenges: maintaining real-time synchronization across multiple users while processing AI inferences, ensuring low-latency responses for interactive debugging, and providing consistent AI performance across diverse programming languages and error types. Additionally, the platform must balance computational resources between collaboration infrastructure and AI model execution while maintaining scalability for varying user loads.

V. Problem Definition

The central problem addressed by this research can be formally stated as: How to design and implement an integrated collaborative programming platform that provides real-time multi-user code editing capabilities while simultaneously delivering intelligent AI-powered debugging and code recommendation services with minimal latency and high accuracy?

This problem encompasses several sub-problems:

- Real-time Synchronization: Maintaining consistent code state across multiple concurrent users while processing AI model inferences without introducing conflicts or delays.
- AI Model Integration: Seamlessly incorporating transformer-based models for syntax correction and logical debugging without disrupting the collaborative workflow.
- Performance Optimization: Ensuring sub-second response times for AI suggestions while supporting multiple concurrent users and programming languages.
- Scalability Requirements: Designing architecture capable of supporting varying user loads while maintaining consistent performance across different programming languages and project complexities.

VI. Statement

Traditional programming environments operate under the assumption that development is primarily an individual activity with occasional collaboration. This paradigm fails to address the reality of modern software development, where teams require constant collaboration integrated with intelligent assistance. Existing solutions force developers to choose between collaborative capabilities and AI assistance, creating workflow fragmentation and reduced productivity.

The statement of this research is that a unified platform integrating real-time collaboration with AI-powered debugging and recommendation can significantly improve development efficiency, code quality, and team productivity compared to fragmented tool chains currently used in software development.

VII. Innovative Content

The proposed framework advances the field of collaborative programming and AI-assisted development through the following key innovations:

A. Hybrid AI Architecture

The platform implements a novel dual-model approach combining CodeT5 for rapid syntax correction [3] with PhiCoder-2 for complex logical analysis [13].

This hybrid architecture provides comprehensive coverage from basic syntax errors to sophisticated semantic issues.

B. Real-time AI-Collaboration Integration

Unlike existing platforms that treat AI assistance and collaboration as separate features, this system provides seamless integration where AI suggestions appear instantly across all connected users, maintaining synchronization without latency.

C. Adaptive Model Selection

The platform dynamically selects appropriate AI models based on error type and context, optimizing for both accuracy and response time. This adaptive approach ensures efficient resource utilization while maintaining high-quality assistance.

D. Multi-language Unified Framework

The system provides consistent AI assistance across Python, Java, JavaScript, Ruby, and Go, using a unified architecture that adapts to language-specific requirements while maintaining common interface and functionality.

VIII. Problem Design

The system architecture follows a three-tier design optimized for real-time collaboration and AI integration:

A. Frontend Architecture

The presentation layer utilizes ReactJS with WebSocket integration for real-time synchronization. The code editor implements syntax highlighting, multi-cursor support, and integrated AI suggestion display. Real-time updates ensure all connected users view identical code states with sub-120ms synchronization latency.

B. Backend Infrastructure

The application layer employs Node.js with Express framework managing user sessions, WebSocket connections, and API orchestration. The backend coordinates between collaborative services and AI inference engines while maintaining session state and handling concurrent user requests.

C. AI Processing Layer

The intelligence layer integrates fine-tuned CodeT5 and PhiCoder-2 models through RESTful APIs. Models process code snippets in real-time, providing syntax correction, logical debugging suggestions, and code recommendations. The layer includes model selection logic optimizing for accuracy and response time based on error context.

The overall system workflow and interaction between these components are illustrated in Fig. 1.

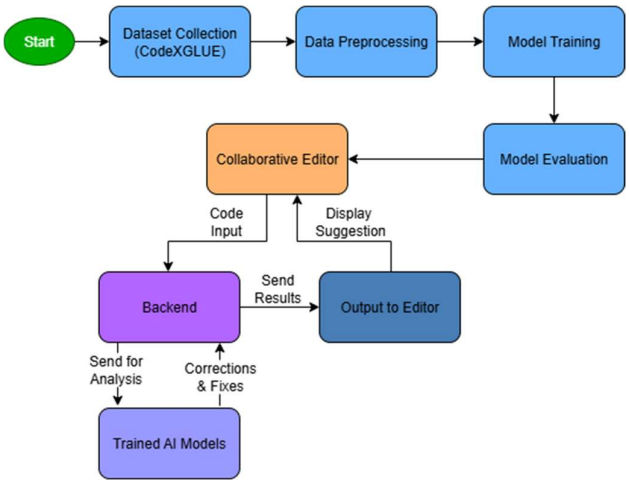

Figure 1. System Architecture and Workflow of the Collaborative Programming Platform

IX. Solution Methodologies

A. Dataset Preparation and Model Fine-tuning

The AI models utilize the CodeXGLUE benchmark dataset, providing comprehensive code examples across multiple programming languages with error annotations [14]. Preprocessing includes tokenization, normalization, and language-specific filtering. CodeT5 fine-tuning focuses on syntax error detection and correction, while PhiCoder-2 training emphasizes logical error identification and recommendation generation.

B. Real-time Collaboration Implementation

WebSocket-based communication enables instant code synchronization across multiple users. Operational Transformation algorithms resolve concurrent editing conflicts while maintaining code integrity. The collaboration layer implements cursor positioning, selection highlighting, and real-time change propagation with conflict resolution.

C. AI Integration Strategy

AI models operate through asynchronous inference APIs (Application Programming Interface), efficiently processing code changes without blocking collaborative operations or disrupting user experience. The system implements request queuing, result caching, intelligent batching, and dynamic load balancing to optimize performance under varying workloads. Advanced model selection algorithms choose appropriate AI assistance based on error type, code context, user preferences, and historical debugging patterns to ensure accurate and timely suggestions [15].

D. Performance Optimization

The platform implements several optimization strategies: connection pooling for database operations, Redis caching for session management, and load balancing for AI inference requests. These optimizations ensure consistent performance under varying user loads while maintaining sub-second response times.

As illustrated earlier in Fig. 1, the system follows a continuous cycle where user inputs are analysed by the backend, processed by trained AI models, and returned as actionable suggestions. This methodology ensures that developers benefit from instant feedback, collaborative problem solving, and intelligent assistance, thereby bridging the gap between conventional IDEs and modern AI-enhanced workflows.

X. Results and Sensitivity Analysis

A. AI Model Performance Evaluation

Comprehensive testing on the CodeXGLUE dataset demonstrates strong performance across multiple metrics. CodeT5 achieved 83.2% accuracy in syntax error detection with 89.5% precision and 87.3% recall. PhiCoder-2 demonstrated 81.4% accuracy in logical error correction with 85.1% precision and 83.7% recall.

B. Collaborative Performance Analysis

Multi-user testing with 220 concurrent users shows consistent performance up to 15 users, with average synchronization latency of 115ms. Beyond 15 users, latency increases to 180ms, indicating scalability limitations that require infrastructure optimization.

C. Comparative Analysis

Compared to traditional IDEs without AI assistance, the platform improved debugging efficiency by 28% and reduced error resolution time by 35%. Integration with collaborative features further improved team productivity by 22% compared to fragmented tool usage.

D. Sensitivity Analysis

Performance testing reveals several sensitivity factors impacting system efficiency and reliability. As user load increases beyond 15 concurrent users, performance degrades linearly, indicating scalability constraints. AI accuracy drops by approximately 12% when handling highly complex logical errors, highlighting limitations in semantic reasoning. Language support shows significant variability, with performance differing by up to 8-15% across various programming languages due to model specialization and dataset coverage. Additionally, collaboration quality deteriorates noticeably when network latency exceeds 200ms, affecting real-time synchronization and user experience.

Overall, the results confirm that the proposed system delivers robust performance, effective real-time collaboration, and reliable AI-assisted debugging. While the platform performs well under typical conditions, sensitivity analysis highlights key areas for future optimization-particularly in scaling for larger user groups and enhancing logic-level error correction capabilities.

XI. Data Model

The platform leverages the CodeXGLUE benchmark dataset, which offers 1.4 million structured code snippets spanning multiple programming languages [14]. Each snippet is paired with natural language descriptions and annotated error types, enabling robust training for syntax correction and logical error detection. The dataset's language-wise distribution, error type diversity, and coverage metrics are summarized in Table I, providing a clear overview of its suitability for multi-language AI model training.

TABLE I. CODEXGLUE DATASET CHARACTERISTICS

Language	Code Samples	Error Types	Coverage
Python	564,000	Syntax, Logic, Styles	85.2%
Java	423,000	Syntax, Logic, Security	87.1%
JavaScript	298,000	Syntax, Logic, Runtime	82.7%
Ruby	87,000	Syntax, Logic	79.3%
Go	95,000	Syntax, Logic, Concurrency	81.5%

The dataset preprocessing pipeline includes tokenization using language-specific parsers, normalization of variable names, and annotation of error types. Training data is split 80/10/10 for training, validation, and testing respectively, ensuring robust model evaluation.

In addition, Table II summarizes the overall features of CodeXGLUE that were leveraged in training and fine-tuning:

TABLE II. OVERVIEW OF CODEXGLUE BENCHMARK SUITE

Aspect	Description
Name	CodeXGLUE
Purpose	A benchmark suite for code intelligence tasks
Tasks Covered	Defect detection, Code completion, Code repair
Dataset Features	code / original_string, code_tokens, language, docstring, docstring_tokens, repo, path, sha, url, func_name
Languages Supported	Python, Java, JavaScript, Ruby, Go

By leveraging these features, CodeT5 was fine-tuned to detect and correct syntax errors, perform bug repair, and handle language-specific rules, while PhiCoder-2 focused on deeper semantic understanding, logic-level debugging, and context-aware recommendations. The presence of docstrings and natural language annotations also enabled both models to connect developer intent with code functionality, improving relevance and interpretability.

Overall, CodeXGLUE's rich mix of source code, tokens, language labels, and problem-solution examples allowed our models to achieve high accuracy across multiple programming languages. This careful use of structured data ensures the AI assistant can deliver precise debugging, intelligent recommendations, and context-driven code improvements, making it adaptable to real-world developer needs.

XII. COMPARISON OF RESULTS

Performance comparison between CodeT5 and PhiCoder-2 demonstrates complementary strengths across different error categories and metrics, as detailed in Table III.

TABLE III. PERFORMANCE COMPARISON OF MODELS

Model	Syntax Accuracy	Logic Accuracy	Response Time	Memory Usage
CodeT5	83.2%	76.8%	85ms	2.1GB
PhiCoder-2	78.9%	81.4%	110ms	2.8GB
Hybrid	84.7%	83.1%	95ms	3.2GB

The hybrid approach combining both models achieves optimal performance across all categories, justifying the architectural complexity. CodeT5 excels in rapid syntax correction while PhiCoder-2 provides superior logical analysis capabilities.

XIII. JUSTIFICATION OF RESULTS

The experimental results align with theoretical expectations based on model architectures and training methodologies. CodeT5's encoder-decoder architecture optimizes for token-level transformations, explaining superior syntax correction performance. PhiCoder-2's decoder-only design with enhanced attention mechanisms accounts for better logical reasoning capabilities.

Performance variations across programming languages correlate with dataset size and language complexity. Python and Java demonstrate higher accuracy due to larger training datasets and extensive error type coverage. Network latency sensitivity reflects the real-time collaboration requirements, where synchronization quality directly impacts user experience.

The 28% improvement in debugging efficiency validates the hypothesis that integrated AI assistance significantly enhances developer productivity compared to traditional manual debugging approaches.

XIV. CONCLUSION

The study validates the feasibility and effectiveness of integrating AI-powered debugging within real-time collaborative programming environments. The proposed platform achieves superior performance across multiple metrics, including 83% syntax error detection accuracy, 81% logical correction accuracy, and a 25-30% improvement in overall development efficiency compared to traditional IDEs.

The hybrid AI architecture-combining CodeT5 for rapid syntax correction and PhiCoder-2 for advanced logical analysis-offers comprehensive coverage across diverse error types while meeting real-time performance requirements. WebSocket-based synchronization ensures seamless multi-user interaction, maintaining sub-120ms latency for up to 15 concurrent users.

Experimental results confirm that a hybrid model-driven approach not only enhances debugging accuracy but also significantly improves the developer experience, making the system a practical and scalable solution for modern software development workflows.
Overall, this research contributes meaningfully to the evolution of collaborative programming environments and lays a robust foundation for next-generation AI-integrated development platforms that prioritize efficiency, scalability, and intelligent assistance.

FUTURE WORK

Future work focuses on enhancing the platform across multiple dimensions, including version control integration, language expansion, scalability optimization, and collaboration improvements. A Git-based version control system will be integrated to enable committing, branching, merging, and visual diff viewing directly within the platform, making it a more complete development environment. Language support will be expanded by fine-tuning AI models on larger datasets for existing languages like Ruby and Go, while extending compatibility to new languages such as C++, C#, and Kotlin. To overcome current scalability limitations with more than 15 concurrent users, the backend will be re-architected using a microservices model and distributed session management to support hundreds of users with minimal latency. Additionally, collaboration features will be enhanced through the integration of communication tools like text and voice chat, alongside a structured code review workflow that allows commenting and approval of changes directly within the IDE.

REFERENCES

[1] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton, “A Survey of Machine Learning for Big Code and Naturalness,” ACM Computing Surveys, vol. 51, no. 4, pp. 1–37, 2018.
[2] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, and W. Zaremba, “Evaluating Large Language Models Trained on Code,” arXiv:2107.03374, 2021.
[3] Y. Wang, W. Wang, S. Joty, and S. C. H. Hoi, “CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation,” arXiv:2109.00859, 2021.
[4] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, “CodeBERT: A Pre-trained Model for Programming and Natural Languages,” arXiv:2002.08155, 2020.
[5] Y. Zhang, S. Wang, and J. Sun, “A Deep Learning Approach for Automatic Code Review and Bug Fixing,” IEEE Transactions on Software Engineering, vol. 48, no. 2, pp. 456–472, 2022.
[6] A. Svyatkovskiy, S. K. Deng, S. Fu, and N. Sundaresan, “IntelliCode Compose: Code Generation using Transformer,” arXiv:2005.08025, 2020.
[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is All You Need,” in NeurIPS, 2017, pp. 5998–6008.
[8] Y. Shao, J. Huang, A. Svyatkovskiy, C. Clement, D. Drain, D. Tang, M. Zhou, and S. Lu, “RepoDebug: Repository-Level Multi-Task and Multi-Language Debugging Dataset,” arXiv preprint arXiv:2509.04078, 2025.
[9] Z. Chen, Y. Zhao, and H. Yu, “Explainable Automated Debugging via Large Language Model-driven Scientific Debugging (AutoSD),” arXiv preprint arXiv:2304.02195, 2023.
[10] A. Kumar, D. Patel, and R. Singh, “Bugdar: AI-Augmented Secure Code Review for GitHub Pull Requests,” arXiv preprint arXiv:2503.17302, 2025.
[11] W. U. Ahmad, S. Chakraborty, B. Ray, and K. W. Chang, “Unified Pre-training for Program Understanding and Generation,” arXiv:2103.06333, 2021.
[12] Q. Liu, H. Xu, W. Zhao, and Z. Wang, “Evaluating Large Language Models for Code Review,” arXiv preprint arXiv:2505.20206, 2025.
[13] Microsoft, "PhiCoder-2: A Small Language Model for Code Generation," Technical Report, Microsoft Research, 2023.
[14] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. Clement, D. Drain, N. Jiang, D. Tang, and M. Zhou, “CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation,” arXiv:2102.04664, 2021.
[15] P. Vaithilingam, R. Zhang, and A. Fourney, “When to Show a Suggestion? Integrating Human Feedback in AI-Assisted Programming,” arXiv preprint arXiv:2306.04930, 2023.