International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 12 | Dec - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

AI-Assisted Code Editors with Real-Time Collaboration: A Comprehensive Review

Narasimha Dixit[1], Aniket Patil[2], Ayaan Shilledar[3] , Krutika Sambranikar[4],Prashant Chavan[5]

[1]*Department of Computer Science(AIML),KLS Vishwanathrao Deshpande Institute Of Technology Haliyal,India*
[2]*Department of Computer Science(AIML),KLS Vishwanathrao Deshpande Institute Of Technology Haliyal,India*
[3]*Department of Computer Science(AIML),KLS Vishwanathrao Deshpande Institute Of Technology Haliyal,India*
[4]*Department of Computer Science(AIML),KLS Vishwanathrao Deshpande Institute Of Technology Haliyal,India*
[5]*Department of Computer Science(AIML),KLS Vishwanathrao Deshpande Institute Of Technology Haliyal,India*

Abstract - The rapid growth of distributed software development requires advanced collaborative coding tools to fulfill market demands. AI-assisted code editors serve as revolutionary platforms which transform developer teamwork through instant code creation capabilities. The editors resolve essential code synchronization and developer productivity and quality assurance challenges through their real-time collaboration system and generative AI capabilities. The development process depends on real-time databases and collaborative frameworks which enable users to edit documents without any difficulties when working with multiple users. The system enables developers to work together through live code synchronization and cursor tracking and integrated chat systems and autosave features which prevent workflow interruptions and conflicts. The AI-assisted code editors achieve their unique status through their ability to merge large language models (LLMs) with generative AI technologies. The systems deliver three essential features which include intelligent code recommendations and immediate syntax error detection with automatic correction and automated documentation creation for intricate functions. The system decreases developer workload while making code more understandable and helping teams maintain professional coding standards. The coding environment allows developers to access AI assistants which function as interactive tools. The real-time assistants help developers solve coding questions and provide optimization recommendations and debugging assistance and brainstorming support. The platforms operate with robust extensible code editor frameworks which allow users to work with multiple programming languages and customize their interface through themes and dynamic file organization for efficient project management. AI-powered collaborative editors unite real-time teamwork with intelligent automation through their design inspiration from code generation and AI-assisted development tools. The tools help teams generate superior code while establishing conditions for AI-assisted innovation to enhance team member collaboration.

Key Words: AI code editor, real-time collaboration, generative-AI, large language models (LLM), smart code suggestions, auto- documentation, syntax error detection, collaborative development tools, developer productivity.

1.INTRODUCTION

This The rapid development of software requires new tools which deliver enhanced efficiency and scalability and enable team collaboration. The AI-assisted code generation technology brings a new innovation which revolutionizes developer code creation and team collaboration methods. The increasing complexity of software systems makes it challenging for developers to manage large codebases while keeping development processes efficient and error- free. AI tools that apply transformer models for code generation solve problems that developers face when working with complex codebases.

The AI-assisted code generation system in IntelliCode Compose uses transformer models to achieve a new level of integration between artificial intelligence and software engineering. The system analyzes large codebases to detect programming patterns while delivering instant applicable code snippets which help developers tackle complicated tasks instead of writing code manually. The tools enable distributed teams to enhance their productivity and collaboration because software development now takes place between teams located in different parts of the world. Scientists actively develop AI-assisted code generation through transformer models and NLP and deep learning algorithms to create more accurate and efficient code generation systems. The research focuses on three essential objectives which include enhancing code completion systems and building error detection tools and developing workflow-based suggestions that match developer needs. The popularity of real-time collaboration tools in coding platforms continues to rise because they enable developers to perform seamless code editing and review tasks together. The paper performs an in-depth assessment of AI-assisted code generation research by studying IntelliCode Compose in detail.The research examines 44 reference papers to deliver a thorough overview of this developing field's progress and obstacles and potential future developments. The paper establishes connections between

these findings with current research on AI-Assisted Code Editor with Real-Time Collaboration that investigates practical applications of these technologies in actual coding environments. layout of the typeset paper will not match this template layout.

2. Methodology

The research followed a structured method to evaluate AI code generation through transformer models in academic and technical contexts. The research paper IntelliCode Compose: Code Generation Using Transformer served as the primary source to study both theoretical and practical aspects of this field.

The research methodology follows this structure:

- Selection of Primary Source:

The survey begins with IntelliCode Compose as its main paper because this research paper holds significant influence in code generation through transformer models. The paper receives numerous citations because it demonstrates modern AI-assisted programming tool development directions. It also aligns closely with the technical features and goals of the project AI-Assisted Code Editor with Real-Time Collaboration.

- Compilation of Reference Literature:

The IntelliCode Compose paper included 44 references which were used for evaluation purposes. The references present fundamental concepts alongside recent developments and technical breakthroughs in transformer models and code completion engines and neural language models and developer tooling and software engineering best practices. The research team accessed all papers by searching through digital academic databases which included IEEE Xplore and ACM Digital Library and arXiv and Google Scholar.

•Categorization of Literature:
•The research papers received categorization to enable efficient analysis of the data.
•The main categories consist of:
•Better Transformer adaptations and architectures for code generation.
•Optimized Learning-based models for code completion.
•Code summarization, detection and documentation generation.
•Real-time collaborative development assistance tools.
•Evaluation frameworks for AI-based coding systems.In-depth Review and Note-taking.

The evaluation process of each paper exposed its research objectives and its methods and algorithms and datasets and performance assessment criteria and ultimate research results. The notes presented a brief overview of the innovation along with the advantages and weaknesses of each research study. The assessment analyzed papers which developed innovative transformer models and papers that studied code generation context understanding.

- Comparative Analysis with AI Assisted Code Editor:

The literature analysis enabled researchers to conduct a comparative assessment of the AI Assisted Code Editor project. The evaluation process examined these specific aspects:

Functional overlap (e.g., code suggestion, syntax awareness)
oReal-time collaboration support.
oScalability and integration features.
oCustomization and user interaction design.
oThe evaluation of AI Assisted Code Editor's uniqueness and relevance to the broader research ecosystem occurred through this step.

- Documentation:

The results were documented through a structured process which followed the sequence of background information followed by review and comparison and then conclusion. The survey design follows a methodological approach which enables reproduction and maintains academic standards and full transparency.

2.1 Modeling And Analysis

The fundamental principle of AI-assisted code generation depends on advanced machine learning models which use transformer-based models to understand and generate code suggestions. The following section explains the models from research papers focusing on IntelliCode Compose while demonstrating their differences with the methods and design structure of this project.

2.1.1 Transformer Models for Code Generation:

State-of-the-art code generation systems depend on transformer models because these models demonstrate superior ability to analyze both sequence order and semantic patterns in code. The transformer architecture in IntelliCode Compose enables Microsoft Visual Studio to generate context-based code completion suggestions. The models learn programming idioms and syntax rules and structural code patterns through training on large open-source GitHub repository datasets.

2.1.2 Dataset and Pretraining:

State-of-the-art code generation systems depend on transformer models because these models demonstrate exceptional ability to recognize both sequence order and semantic relationships in code. The transformer architecture in IntelliCode Compose enables Microsoft Visual Studio to generate context-based code completion suggestions. The models learn programming idioms and syntax rules and structural code patterns through training on large open- source GitHub repository datasets.

2.1.3 Model Optimization Techniques:

The research literature demonstrates multiple model optimization approaches which enhance performance through Masked Language Modeling (MLM) for partial

code snippet understanding and Autoregressive generation for sequential code suggestion. The model requires domain-specific dataset fine-tuning for Java and Python project development. The generated code quality and diversity improve through the implementation of Beam search and top-k sampling techniques.

2.1.4 Real-time Code Assistance and IntelliCode Compose:

The IntelliCode Compose tool lets users create code through interactive functions that operate within their Integrated Development Environment (IDE). The modeling system combines transformer predictive functions with user code-specific context information. The system produces recommendations that match the active line while it analyzes both the surrounding code and the complete function structure. The system performs real-time inference through three essential elements which include Lightweight model deployment. The system extracts code information through contextual embeddings that analyze the surrounding programming code. The system sends fast API requests to local or cloud-based models to retrieve suggestion results.

2.1.5 SynapseCode System Architecture :

SynapseCode provides intelligent code suggestions alongside real- time collaboration features as its main focus. The system uses three main components to achieve its functionality: An embedded transformer model (such as CodeT5 or CodeBERT) for local code understanding. WebSocket or Firebase-based backend for real-time communication and collaboration. Frontend editor (e.g., based on Monaco Editor) with integration hooks for suggestion APIs.User- aware assistance, where suggestions are personalized based on the editing behavior of collaborators.

2.1.6 Analysis and Comparison:

Aspects	**Traditional Code Editors**	**AI-Assisted Collaborative Code Editor**
Code Suggestions	Basic autocompletion	Smart code and context aware suggestions
Documentation	Manual or plugin-based	Auto-generated docs and code explanation
Error Handling	Syntax checking	Real-time error detection
Collaboration	No real-time teamwork support	Multi-user editing with live updates

Table -1: Comparison

2.1.7 Analysis and Comparison:

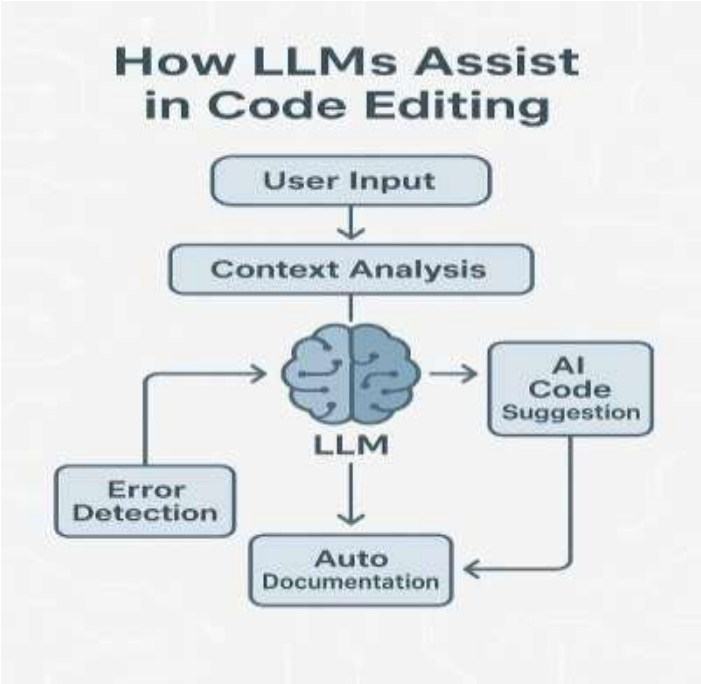

Fig.1. How LLMs Assist in Code Editing.

The AI Assisted Code Editor system provides intelligent code suggestions together with real-time collaborative features. The system architecture consists of three main components which include: A transformer model (CodeT5 or CodeBERT) operates as an embedded system to analyze local code structure. The system uses WebSocket or Firebase-based backend technology to enable real-time communication and collaborative work. The system uses Monaco Editor as its frontend editor through which users can access suggestion APIs. The system provides user-aware assistance through personalized suggestions which depend on how users edit the code during collaborative work. The system provides two main features through its architecture which includes intelligent code suggestions and real-time collaborative functionality. The system architecture consists of three main components which include: A transformer model (CodeT5 or Code BERT) operates as an embedded system to analyze local code structure. The system uses WebSocket or Firebase-based backend technology to enable real- time communication and collaborative work. The system uses Monaco Editor as its frontend editor through which users can access suggestion APIs. The system provides user-aware assistance through personalized suggestions which depend on how users edit the code during collaborative work.

The evaluation shows IntelliCode Compose provides outstanding intelligent completion tools through its fully developed IDE system yet AI Assisted Code Editor expands these concepts by building a team-based coding environment which solves issues related to shared code understanding and team member communication.

3. Background And Motivation

The programming environment undergoes a transformation through Artificial Intelligence (AI) integration in software development because developers now use AI to generate code and perform debugging and optimization tasks. The increasing complexity of contemporary software development requires developers to use tools which help them sustain code excellence while enhancing their work efficiency and minimizing monotonous activities. The development of AI-assisted code generation tools has triggered substantial research efforts which produced intelligent systems that identify and predict programming patterns.

The field has achieved its most significant progress through the implementation of transformer-based systems for code modeling. The programming language understanding success of NLP transformer models led to the development of GPT and CodeBERT and T5 for programming language understanding. The models process extensive code databases to execute multiple programming tasks including code completion and summarization and translation and bug detection. IntelliCode Compose operates as a functional model of these systems because it implements them within

Microsoft Visual Studio to deliver real-time context-specific code recommendations.The existing tools show advancement yet they do not fulfill the needs of collaborative coding platforms. AI-enhanced code editors continue to optimize developer performance independently instead of supporting simultaneous real-time teamwork between multiple developers. The lack of appropriate tools led to the development of AI Assisted Code Editor which unites intelligent code completion with immediate team collaboration features.

The AI Assisted Code Editor enhances team work through transformer models operating in a dynamic

Fig 2. Evolution of Coding Tools.

editing environment to improve collaboration and code understanding between team members.

The research study examines IntelliCode Compose and other AI-assisted systems through their existing literature to evaluate their benefits and drawbacks and identify the key factors that drove the creation of AI Assisted Code Editor. The research examines transformer-based models in coding environments to understand present developments and potential future applications of intelligent collaborative programming tools.

3.1 Gaps In Current Research

AI-assisted code generation has achieved notable advancements yet researchers can explore new opportunities through IntelliCode Compose development because current systems have specific weaknesses. The existing gaps in current systems demonstrate their operational restrictions which motivate developers to create more dependable systems that understand context and work together.

3.1.1 Limited Real-Time Collaboration Support:

The primary operation of IntelliCode Compose and other present-day AI code generation tools functions independently within user-specific environments. The current tools do not support real-time collaboration because this feature remains vital for contemporary team-based software development. The increasing requirement for AI-based solutions that support collaborative code editing and immediate communication and conflict resolution stems from the expanding use of remote and distributed work arrangements.

3.1.2 Limited Context Understanding:

The IntelliCode Compose tool produces recommendations through analysis of local code structure yet it lacks understanding of project-level information. The current models fail to recognize file and class and module dependencies which leads to insufficient or wrong recommendations when developers work on complex projects.

3.1.3 Language and Framework Bias:

The training data for most transformer-based code generation models comes from open-source repositories which show biased

distributions of programming languages and frameworks. The models achieve better results when processing Python, JavaScript and Java code.

3.1.4 Lack of Personalization:

The present systems lack the ability to modify their operations based on individual coding approaches and team-defined standards. The current models fail to detect developer-specific preferences and architectural designs and formatting rules which developers use. The field requires research to understand how AI code assistants should learn from user behavior through dynamic personalization systems.

3.1.5 Privacy and Security Concerns:

AI models trained on public codebases produce unintentional code suggestions which contain security vulnerabilities and licensing problems and sensitive patterns. The process of real-time inference through cloud-based models reveals proprietary code to outside servers which generates security risks. Privacy-preserving models require secure operation within local or hybrid systems.

3.1.6 Evaluation Metrics for Code Quality:

Research into AI-generated code effectiveness assessment continues to develop as a new field of study. The current evaluation metrics which use BLEU scores and token-level accuracy do not effectively measure semantic correctness and maintainability and performance. Real-world code generation systems need new evaluation frameworks which will assess their operational value effectively.

3.2 Future Scope

The AI-assisted code generation tools including IntelliCode Compose demonstrate how artificial intelligence can transform software development through their shown capabilities. The field exists at its beginning stage while providing various opportunities to enhance its development. The research and development potential in this domain extends widely into promising territory.

3.2.1 Expansion into Real-Time Collaborative Development:

AI assistance will evolve into a future development path which will integrate with actual-time collaborative coding platforms. The AI Assisted Code Editor proposes that AI technology can create smart developer teamwork between multiple users while providing help to individual developers. AI-based real-time conflict resolution systems combined with shared code suggestions and synchronized team workflows will revolutionize the way distributed software teams work together.

3.2.2 Project-Wide and Cross-Project Context Understanding:

Future AI systems need to advance their current ability to analyze limited code snippets. AI systems need to understand complete project structures and all file relationships and past project records to generate intelligent code suggestions that match project context. AI systems will enhance their ability to support complex software development through cross-project learning which enables them to learn from multiple connected projects.

3.2.3 Personalized and Adaptive AI Models:

The market demand for AI tools that modify their operations based on developer actions and team coding rules and

organizational standards keeps growing. Research should focus on building models which learn from ongoing user interactions to generate customized coding assistance that fits individual developer and team preferences and workflows.

3.2.3 Privacy-Preserving AI Systems:

The growing dependence on AI tools will make data protection and user confidentiality issues more urgent than ever. The development of future AI systems needs to focus on building secure local AI models which operate independently from external servers without exposing proprietary code.

3.2.4 Integration with Modern Software Engineering Tools:

The implementation of AI-assisted code generation should occur seamlessly within current software engineering toolchains which include version control systems and CI/CD pipelines and code review platforms and testing frameworks. A single AI assistant that handles all stages of software development from coding to deployment and maintenance operations would significantly boost developer efficiency.

3.2.5 Improved Evaluation Metrics and Benchmarking:

Research in the future should focus on developing enhanced evaluation metrics which assess both syntactic accuracy and semantic quality and operational efficiency and security and maintainability of automatically generated code. The development of standardized benchmarking datasets and protocols for AI code generation models will enable researchers to evaluate new advancements through consistent and meaningful assessments.

3. CONCLUSION

The survey of IntelliCode Compose and other AI-assisted code generation tools through literature analysis demonstrates artificial intelligence has brought major transformations to software development practices. The tools that use transformer-based models for intelligent code completion have demonstrated their ability to enhance coding speed and reduce manual tasks which results in better developer productivity. The tools demonstrate an AI-based development environment which provides developers with automated code suggestions that match their current work environment. The tools function independently within single files yet they fail to address complex project structures and individual requirements and simultaneous team work. The ethical deployment of AI-generated code requires developers to follow responsible AI development practices because of security risks.

Artificial intelligence shows promise for software development but IntelliCode Compose's intelligent coding assistance has not reached its full potential. Research must focus on solving existing challenges by improving context understanding and personalized recommendations and secure model deployment and creating better assessment methods. The software engineering lifecycle will experience a complete transformation through AI-assisted tools which will evolve from basic code completion features into vital partners for all development stages. The upcoming years will bring revolutionary changes to software development and maintenance through continuous innovation and refinement of AI-assisted tools.

ACKNOWLEDGEMENT

We thank Dr. V. A. Kulkarni who serves as Principal of KLS Vishwanathrao Deshpande Institute of Technology Haliyal for allowing us to use their facilities and work environment which enabled us to finish this review paper. We thank Prof. Poornima Raikar who leads the Computer Science (AI & ML) department for her ongoing support and her helpful advice and her continuous backing throughout our work. throughout the course of this work. We owe our deepest gratitude to Prof. Narasimha Dixit who provided ongoing assistance through his expert guidance and valuable recommendations that led to our successful paper completion. His ongoing feedback and guidance enabled us to develop our knowledge and methods for this research.

REFERENCES

[1] Z. Austin, L. Smolensky, Y. Liu, R. Wray, C. Brooks, S. Fink, and A.Sutton, "IntelliCode Compose: Code Generation Using Transformer," In Proceedings of the 43rd International Conference on Software Engineering (ICSE), 2021.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.
N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention Is All You Need," In Advances in Neural Information Processing Systems (NeurIPS), 2017.

[3] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. Pinto, J. Kaplan,
H. Edwards, et al., "Evaluating Large Language Models Trained on Code," • arXiv preprint arXiv:2107.03374, 2021.

[4] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, and D. Jiang, "CodeBERT: A Pre- Trained Model for Programming and Natural Languages," arXiv preprint arXiv:2002.08155, 2020.

[5] A. Svyatkovskiy, S. Deng, S. Fu, and N. Sundaresan, "Intellicode: Leveraging AI to Improve Developer Productivity," In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), 2020.

[6] W. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to Represent Programs with Graphs," In International Conference on Learning Representations (ICLR), 2018.

[7] P. Yin and G. Neubig, "TRANX: A Transition-Based Neural Abstract Syntax Parser for Semantic Parsing and Code Generation," In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2018.

[8]] A. Rabinovich, M. Stern, and D. Klein, "Abstract Syntax Networks for Code Generation and Semantic Parsing," In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL), 2017.

[9] M. Brockschmidt, "Generative Code Modeling with Graphs," arXiv preprint arXiv:1905.13340, 2019.

[10] W. U. Ahmad, S. Chakraborty, B. Ray, and K. Chang, "Unified Pre-training for Program Understanding and Generation," • arXiv preprint arXiv:2103.06333, 2021.