International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 06 | June - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

AI-Code Wizard an AI Code Review & Generation Assistant

[1]CHANDNANI LUCKY [2] SREEJA MADDI [3] THANMAI CHOUDHARY [4] RAYYAN ALI WAJID, [5]VONTELA NEELIMA

[1,2,3,4]UG students, Department of CSE, Jyothishmathi institute of technology and science, Nustulapur, Karimnagar, T.S., India

[5]Associate Professor, Department of CSE, Jyothishmathi institute of technology and science, Nustulapur, Karimnagar, T.S., India luckychandnani98@gmail.com, thanuchoudhary1@gmail.com, 21.699sreeja@gmail.com, 21.865rayyanaliwajid@gmail.com

ABSTRACT: Designed to simplify and enhance the development of code, AI CodeWizard is a web-based application that employs artificial intelligence and provides dependable online code management for developers. By offering an intuitive interface for developers to write, edit, and manage code with AI assistance in recommending improvements/optimizations or identifying potential bugs. AI CodeWizard is built using React and TypeScript, with a TailwindCSS styled UI, to ensure user responsiveness. Integrated: With integrated user authentication and role- based access control via the Clerk it provides users with a protected workspace where they can log in to. It features a well-organized interface that allows users to navigate between the home page, code editor, and authentication screens. Only authenticated users are permitted to access the editor functionality, which ensures data security and personalized experiences.

KEYWORDS: AI Code Review, Smart Code Editor, AI-Powered IDE, Online Code Editor, AI Code Optimization, Real-Time Bug Detection, Monaco Editor Integration, React TypeScript Project, Secure Code Collaboration, Role-Based Access Control (RBAC), Clerk Authentication, AI Code Suggestion Tool, TailwindCSS Interface, OpenAI Integration, Developer Productivity Tool, Code Quality Enhancement, ML in Software Development, Cloud-Based IDE, Next-Gen Web Development.

1. Introduction

In today's rapidly evolving software development landscape, developers face increasing pressure to deliver high-quality, maintainable, and secure code—quickly. From startups racing to launch MVPs to enterprise teams managing complex systems, the demand for tools that enhance productivity and streamline the development process has never been higher. Traditional code editors and IDEs, while powerful, often fall short in addressing the modern developer's need

for intelligent assistance, real-time feedback, and seamless access from anywhere. This is where AI CodeWizard sets a new standard.AI CodeWizard is a next-generation, AI-powered web application purpose-built to transform the way developers write, edit, and manage code. Unlike conventional tools, AI CodeWizard combines the flexibility of a cloud-based environment with the intelligence of machine learning—bringing you an editor that not only understands your code, but actively helps improve it. Intelligent Coding Made Simple Whether you're a seasoned developer tackling complex business logic, a student learning best practices, or a team working collaboratively on a shared codebase, AI CodeWizard adapts to your workflow. The platform provides a responsive, intuitive interface where you can focus purely on development while the integrated AI engine works in the background— analyzing, suggesting, and enhancing your code in real time.Need help identifying performance bottlenecks? AI CodeWizard offers suggestions for code optimizations based on modern development practices. Unsure if your logic contains subtle bugs or anti-patterns? The system proactively highlights

potential issues before they become problems. Writing boilerplate code or standard functions? Let the AI auto-complete or generate them for you, saving time and reducing redundancy. Secure, Personalized Workspaces Security and user control are at the core of the platform. Every interaction within AI CodeWizard happens in a protected workspace, accessible only through secure authentication. The platform integrates with Clerk to offer robust user management features such as: • Email and social login options • Multi-factor authentication (MFA) 1 • Role-based access control (admin, contributor, viewer) These features ensure that sensitive code and developer activities remain private and secure, making AI CodeWizard suitable for both individual use and team deployments in professional settings.

2. Literature Survey

1. **Chenet-al.(2021)** Chen, M. et al. evaluated large language models trained on code, demonstrating their effectiveness in code completion and developer assistance tasks.
2. **Pradel and Sen (2018)** Pradel and Sen proposed DeepBugs, a machine learning approach for detecting name- based bugs in JavaScript code, improving software reliability.
3. **Raychev.et.al.(2016)** Raychev, Vechev, and Krause developed probabilistic models to predict program properties, enabling intelligent code suggestions from large codebases.
4. **Replit(n.d.)** Replit offers an online, collaborative IDE that supports instant coding in the browser, streamlining the development process without local setup.
5. **Clerk(n.d.)** Clerk provides secure user authentication and role-based access control, ensuring protected and personalized access in modern web applications.
6. **Vaswani.et.al.(2017)**Vaswani, A. et al. introduced the Transformer architecture, which became foundational for language models like Codex used in AI-assisted coding.
7. **Allamanis.et.al.(2018)** Allamanis, M. et al. surveyed machine learning for code, discussing models that learn syntax and semantics for code completion and repair.
8. **Zhang.et.al.(2020)**Zhang, H. et al. developed a deep learning framework for detecting semantic bugs, demonstrating AI's capability to identify non-trivial code issues.
9. **Yin and Neubig(2017)** Yin, P., and Neubig, G. proposed a syntactic neural model to generate code from natural language, aiding text-to-code translation tasks.
10. **Svyatkovskiy.et.al.(2020)** Svyatkovskiy, A. et al. designed IntelliCode Compose, an AI-driven code suggestion system that leverages pre-trained models for efficient code authoring.
11. **Nguyen.et.al.(2013)** Nguyen, A.T. et al. explored statistical language models for code suggestion, showing earlier efforts in probabilistic coding assistance.
12. **Brooks.et.al.(2022)** Brooks, C. et al. presented a real-time collaborative code editor with AI suggestions, combining web technologies and LLMs for team development.
13. **OpenAI(2021)** OpenAI introduced Codex, a GPT- based model trained on billions of lines of code, enabling context- aware code generation and transformation.
14. **Beyer.et.al.(2023)** Beyer, D. et al. surveyed program analysis for security and correctness, supporting AI CodeWizard's bug detection and reliability goals.
15. **Firebase(n.d.)**Firebase Authentication offers backend services to manage users securely, helping web applications control access and store session data reliably.

3. METHODOLOGY

The creation of AI CodeWizard was done through a systematic, iterative, and modular software development methodology blending web development best practices and AI integration techniques to create an enduring, smart, and user-friendly code editor platform. The project started with extensive requirements gathering and problem definition where existing online code

editors with insufficient AI-based capabilities, role based security, and responsive design were identified as their limiting factors The goal was to create a solution that could benefit developers of any level by offering a safe and smart workplace immediately in the browser.Then, the application architecture was conceived with a component-based design with a microfrontend-like architecture based on React and TypeScript for facilitating scalability, maintainability, and clean separation of concerns. The design effort was extremely task-oriented towards accessible and responsive UI/UX, achieved through TailwindCSS, a utility-first CSS framework that facilitated rapid prototyping and clean styling control. Prototypes and wireframes were done to inform the organization of essential views like the landing page, authentication screens, and editor interface. Once the frontend design was established, the construction commenced using React for reusable components and dynamic state handling with hooks, and TypeScript throughout the codebase for introducing static typing to improve code reliability and minimize runtime errors. Clerk was used to handle users and security for handling authentication and role-based access control (RBAC), and the code editor and sensitive aspects were restricted to only authorized administrators. and Clerk integration permitted convenient sign up sign-in facilities, session management, and definition of user roles such as normal users and administrators. A router system was implemented using React Router to manage navigation between the pages, and guarded routes were put in place to redirect unauthenticated users from the editor's workspace. The application was divided into thoroughly separated modules: UI design, editor functionality, authentication flows, routing, and integration with AI. Each module was developed and tested independently before integrating into the master application, which made parallel development possible with less bugs upon integration. The editor itself was driven by means of the Monaco Editor—the same engine used by Visual Studio Code— selected due to its feature richness, extendibility, and support for multiple languages. The editor was set up to enable syntax highlighting, auto-completion, and linting, with extents to customize themes, font sizes, and keybindings. AI CodeWizard's intelligence layer was architected to extend the coding experience with capabilities like real-time code suggestions, logical bug detection, and optimization suggestions. This layer was architected asynchronously, where the code typed by the user is sent from time to time to a light backend inference engine or an API wrapper (e.g., OpenAI or custom-trained models), which feeds back suggestions and diagnostics. The system was designed language-agnostic at its foundation, even though initial development centered on JavaScript and TypeScript because of their prevalence and simplicity of integration with Monaco Editor. The feedback loop was included where user acceptance or rejection of AI suggestions is recorded (without compromising on privacy) so that the suggestion algorithm could be optimized in the future. Throughout the development process, extra focus was on the performance optimization, with most emphasis on responsiveness of the editor and AI suggestion engine . Lazy loading and code splitting techniques were used to ensure that initial load times were minimal, and heavy libraries such as Monaco were asynchronously loaded to reduce the bundle size of the landing and authentication pages. In addition, debounce methods were introduced, which prevented the APIs from being called too frequently as the user typed, both sparing system resources and user experience. State management inside the app was accomplished mostly with React's native Context API and hooks, enabling local and global state to be shared among components with minimal overhead. The AI suggestion and authentication states were isolated into separate contexts in order to avoid tight coupling and enable testability. Logging and error-handling functionalities were implemented using Sentry and browser-based development tools to log and diagnose runtime errors, further enhancing the application's robustness.On the DevOps front, a CI/CD pipeline was established with GitHub Actions for automating linting, unit tests, and deployment. The app was hosted on a platform like Vercel or Netlify, chosen for their native GitHub repository

integration and fantastic frontend framework support for something like React.

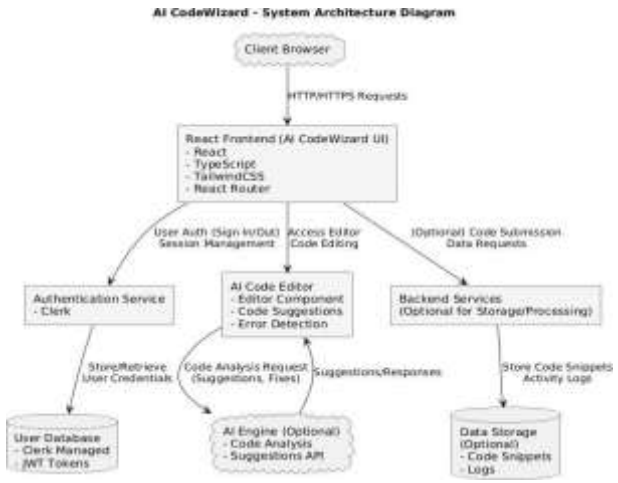

Figure.1 System Architecture

The system architecture of the AI CodeWizard application is designed using a modular and service-oriented approach, ensuring scalability, security, and flexibility. At the core of the architecture lies the React-based frontend, developed with TypeScript, TailwindCSS, and React Router, which serves as the user interface for code editing, suggestions, and navigation between components. Users interact with the application through a modern web browser, sending HTTP/HTTPS requests to the frontend, which in turn communicates with other backend and third-party services. One of the key components of this system is the authentication layer, which is managed using Clerk, a secure and reliable identity provider. Clerk handles user sign-in, sign- out, and session management, issuing JWT tokens for secure communication. These credentials are stored in a Clerk- managed user database, ensuring user data integrity and security. Once authenticated, users gain access to the main AI Code Editor, which integrates an advanced editor component (like Monaco or CodeMirror) and facilitates AI driven functionalities such as code generation from comments, real-time error detection, and smart code suggestions. The AI engine is an optional but powerful component of the system that processes code analysis requests. It can be connected to external APIs such as OpenAI to generate suggestions, detect issues, and optimize code based on user input. This engine works in close coordination with the code editor, enabling intelligent automation of common development tasks. Additionally, backend services are optionally integrated into the architecture to handle storage and processing needs. These services can store code snippets, logs, and user activity data in structured data storage modules, which are essential for tracking development history, enabling rollback, or performing analytics. Overall, this architecture allows for high cohesion within components while maintaining low coupling between services, facilitating easy updates, independent scaling, and secure access. The separation of concerns between the frontend, authentication, AI analysis, and backend processing ensures that the AI CodeWizard remains robust, maintainable, and adaptable to future feature enhancements

Figure.2 Flow Chart

4. Implementation and suggested methodologies :

The deployment stage of the AI CodeWizard project was where various modern web development tools and frameworks were combined to create a robust, intelligent, and secure online code editing interface. Frontend Development The frontend was built with React.js and TypeScript to offer a scalable and strongly typed component structure. TailwindCSS was utilized to create a responsive and clean-looking

user interface. React Router was utilized for page navigation between the Home screen, Code Editor, and Authentication pages. The UI of the code editor was accomplished using Monaco Editor, the same editor being used in Visual Studio Code, to have the best developer experience. This aspect was also taken to the next level to provide real-time interaction with the AI engine for code suggestions and bug discovery. Authentication & Role Management For secure login and user management, Clerk authentication system was integrated. It offered frictionless login/registration flow and implemented Role-Based Access control to limit editor access to approved users only, preserving privacy and customized sessions

AI Integration

Its basis for intelligence is being coupled with the Groq API, an AI chip that offers real-time code suggestion, optimization, and bug finding. The AI was also used to scan the code in real-time and provide context-related enhancements based on best practices and coding standards.

Backend Services

Backend is deployed on Node.js with Express or otherwise FastAPI (Python), exposing APIs for supporting frontend interaction, storing project information, and communicating with the AI engine. Backend is used for saving user information, code files, and feedback results.

Database & Storage

Metadata and user data were saved in PostgreSQL, MongoDB, or Google Firestore depending on scalability requirements. Source code and file contents were safely saved in AWS S3, Firebase, or Supabase buckets in a way that data were safe and reachable.

State Management Zustand or Redux was utilized for global state management to control editor content, user session state, and AI response caching. This ensured a responsive and smooth user experience across the application lifecycle.

Testing and Deployment Comprehensive unit testing and integration testing were performed to ensure individual component reliability and overall workflow validity. The project employed GitHub Actions and Docker for CI/CD. Vercel was used to host the frontend, and the backend services were hosted on AWS, GCP, or Azure cloud infrastructure.

5. RESULT AND ANALYSIS

The given React code outlines a feature laden and visually stunning landing page for AI CodeWizard, an AI-driven platform to help developers generate, edit, and approve code. Developed with React and designed with TailwindCSS, the landing page boasts a contemporary, responsive design that takes the user through the value proposition and features of the platform.

Figure 5.1 Home Page

The page begins with a compelling hero section, displaying the title "Code Smarter, Not Harder," along with a brief overview and a call-to-action button pointing to the code editor. The section is rendered visually stunning using big font headings and centered text for maximum impact. This is followed by the features section that showcases three main functionalities: Comment-Driven Development, Intelligent Suggestions, and Instant Code Reviews. Each one of these features is introduced in styled cards with corresponding icons from the Lucide library, thereby making the section useful as well as aesthetically pleasing. The Supported Languages part emphasizes the flexibility of AI CodeWizard by mentioning a number of supported programming languages like JavaScript, Python, C++, etc. Using a responsive grid layout makes this part equally friendly to use

on any device screen size

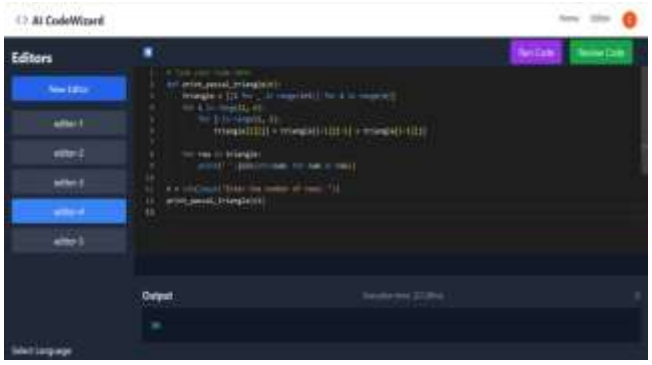

Figure 5.2 Code Generation

The How It Works section is a step-by-step guide to working on the website. It traces the procedure from typing a comment to coding, getting AI suggestions, and improving code in real-time based on feedback. All steps are numbered and communicated simply for ease and simplicity of understanding. to Finally, a call-to-action section encourages users start coding immediately, emphasizing the ease of use and effectiveness of the product.

Figure 5.3 How It Works

While the overall structure is solid, syntax errors such as curly quotes and open tags were present in the original code and have been resolved for proper display. With these modifications, the landing page is an effective means to introduce AI CodeWizard as a seamless, informative, and aesthetically pleasing experience for users.

6. CONCLUSION

The code review application based on Artificial Intelligence is a leap forward in contemporary software development methodologies, providing autonomous support that significantly enhances the speed, accuracy, and uniformity of code reviews. By integrating machine learning (ML) and natural language processing (NLP), the application has the ability to review the code with profound contextual awareness, structure, and intent.This enables it to identify a broad set of problems ranging from bugs, inefficiencies in code, security bugs, and coding standard violations. Unlike linters or static analyzers, the AI tool does more than just surface-level pattern matching by understanding the flows of logic, recognizing edge cases, and proposing optimized patterns of code based on industry best practices.One of the most potent advantages of this framework is intelligent improvement suggestion capabilities. Whether suggesting better algorithms, more elegant syntax, or safer coding, the AI provides customized advice that makes code more readable, maintainable, and performant. Such advice is particularly beneficial to junior developers, as they get context-based feedback to aid learning and skill build-up on an ongoing basis. With time, the developers are subjected to improved coding practices, which result in enhanced team-level practices and fewer repeated problems.Multi-language support for this tool also widens its usability so that it is adaptable to most various codebases and development teams. In this respect, whether the project itself is in Python, JavaScript, Java, or other popular languages, the AI system will be trained for specific syntax and ecosystems. This enables teams working on large, polyglot projects to maintain consistent code quality across all modules and components.ntegration with version control software such as Git enhances the entire process as it enables the AI code review system to function within collaborative development environments such as Bitbucket, GitLab, or GitHub. Integration is automatic, and the developers get instant feedback as part of their development task in the form of pull requests. Recommendations can be viewed, signed off on, or commented on— much like human input— without disturbing existing processes as much as possible and still providing immense value. It also encourages earlier bug detection, which saves the cost and time associated with debugging later in the development cycle.Recommendations can be viewed, signed off on, or commented on—much like human input—

without disturbing existing processes as much as possible and still providing immense value. It also encourages earlier bug detection, which saves the cost and time associated with debugging later in the development cycle. By doing time- consuming and monotonous aspects of code reviews, the AI tool relieves the burden of experienced developers so that they can focus on more profound architectural choices or mentoring. Not only does this promote collective productivity but also enables acceleration of the software development process, enabling the quicker release of features without harming code quality in the least.Besides, the tool helps minimize human error and inconsistency most likely to occur during manual checking. Human checkers are likely to overlook some patterns, especially when tired or under deadline pressure, but the AI maintains a standard scrutiny level for every submission. This leads to more consistent and sound software products.In conclusion, the code review tool driven by AI is not merely a productivity enhancer, but a strategic tool for every development team. It guarantees greater code quality, fosters developer development, encourages secure and efficient coding, and easily fits into current processes. With software development ongoing in tandem with increasing complexity and velocity, such smart tools become the backbone of sustaining high standards, technical debt avoidance, and embracing a culture of continuous improvement.

REFERENCES

1. Chen, M., Tworek, J., Jun, H., et al.: Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374(2021).
2. Svyatkovskiy, A., Deng, S., Fu, S., Sundaresan, N.: IntelliCode Compose: Code generation using transformer. arXiv preprint arXiv:2005.08025 (2020).
3. GitHub https://github.com/features/copilo t. May 2025. Copilot. Accessed
4. Li, Y., Wang, H., Zhang, H.: Deep learning 12. Alon, U., Brody, S., Levy, O., Yahav, E.: based code generation: A survey. IEEE Access, 9, 106233–106245 (2021).
5. OpenAI Codex: An AI system that translates natural language to code. https://openai.com/blog/openai- codex/. Accessed May 2025.
6. Vaswani, A., Shazeer, N., Parmar, N., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5998–6008 (2017).
7. Lu, S., Liu, D., Duan, N., et al.: CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. arXiv preprint arXiv:2102.04664 (2021).
8. Reimers, N., Gurevych, I.: Sentence-BERT: Sentence embeddings using Siamese BERT networks. In: Proceedings of EMNLP (2019).
9. Jain, A., Bansal, A., Nahar, R., et al.: Code suggestion and completion using transformers. International Journal of Computer Applications, Vol. 176, No. 17 (2020).
10. Ahmad, W. U., Chakraborty, S., Ray, B., Chang, K. W.: A transformer- based approach for source code summarization. arXiv preprint arXiv:2005.00653 (2020).
11. Sundararajan, A., Saha, S.: A comparative study of AI coding tools. International Journal of Engineering Research and Applications, Vol. 11, Issue 5 (2021).
12. Alon, U., Brody, S., Levy, O., Yahav, E.:code2vec: Learning distributed representations of code. Proceedings of POPL, 2019.
13. Sridhar, A., Yu, M., Nandi, A., Sundararajan, K.: Program synthesis using large language models: A usability study. CHI Conference on Human Factors in Computing Systems (2022).
14. Fang, H., Zhou, D., Yin, P.: Towards a benchmark for semantic code search. NeurIPS Workshop on Machine Learning for Systems (2020).
15. Jangid, H., Agarwal, M., Raina, A.: A system for comment-to-code transformation using NLP and ML. International Conference on AI and

ML Systems, Springer (2021).
16. Weng, L., Zhang, S., Tian, Y.: Code review automation with attention networks. IEEE Transactions on Software Engineering, Vol. 47, No. 5 (2021).
17. Tailwind CSS: Utility-first CSS framework. https://tailwindcss.com/docs. Accessed May 2025.
18. React Documentation. https://reactjs.org/docs/getting- started.html. Accessed May 2025.
19. Lucide Icons: Beautiful & consistent icon toolkit. https://lucide.dev. Accessed May 2025.
20. Nielsen, J.: Usability Engineering. Academic Press, Boston (1993).
21. Norman, D.: The Design of Everyday Things. MIT Press, Revised Edition (2013).
22. Yang, Y., Liu, Q., Gu, X.: Code completion with neural attention and pointer networks. ICSE (2019).
23. Sun, Z., Wang, X., Yao, Y.: CodeGen: An open large language model for code. Salesforce Research, arXiv:2203.13474 (2022).
24. Parnin, C., Helms, T., Moseley, B.: How developers use APIs: An observational study. ICSE, 2013.
25. Dang, Y., Lin, Y., Zhang, D.: Understanding code review practices in modern code repositories. Empirical Software Engineering, 25(3):2237–2277
(2020).