Volume 10, Issue 4, April – 2025 International Journal of Innovative Science and Research Technology
ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr695

AI Smart Real Time Code Autocompletion, Error Fixing, Code Conversion and Optimization-Fixerbot

Drishya P[1]; Sheerin Farjana M[2]; Jagath M D[3]; Dinesh M[4]

[1]Assistant Professor, CSE Department, SNS College of Engineering, Coimbatore
[2]IV year CSE, SNS College of Engineering, Coimbatore
[3]IV year CSE, SNS College of Engineering, Coimbatore
[4]IV year CSE, SNS College of Engineering, Coimbatore

Publication Date: 2025/04/30

Abstract: As software development continues to evolve, developers are increasingly relying on advanced tools to enhance their coding efficiency. One such tool is the AI-powered autocompletion system, which helps to speed up coding by predicting and suggesting code snippets in real-time. While existing autocompletion tools offer basic suggestions, they often lack accuracy, context-awareness, and the ability to suggest meaningful optimizations. In addition, error detection and fixing, along with performance optimization, are crucial aspects of efficient software development. Developers often spend a significant amount of time identifying and correcting errors in their code, which can be tedious and error-prone. This project proposes an AI Smart Code Autocompletion, Error Fixing, and Optimization tool designed to significantly improve developer productivity by addressing these challenges. The tool leverages advanced machine learning algorithms and natural language processing to offer highly accurate and context-aware code suggestions, reducing the need for manual coding efforts. It goes beyond basic autocompletion by providing error detection and real-time error fixing, helping developers resolve issues before they impact the development process. Moreover, the tool incorporates performance optimization techniques, ensuring that suggested code not only works correctly but is also efficient and optimized for better performance. With features such as intelligent autocompletion, automatic error fixing, and optimization suggestions, this tool streamlines the coding process and enhances software development productivity. The system supports multiple programming languages, offering flexibility and adaptability for developers working across different coding environments. It also provides a comprehensive analysis of the code, ensuring logical consistency and robustness while optimizing performance. By reducing coding time, minimizing errors, and improving the efficiency of the code, this AI-powered tool enables developers to focus on more critical tasks, resulting in a more effective and productive development cycle.

Keywords: *AI-Powered Autocompletion, Code Optimization, Error Fixing, Machine Learning, Software Development Productivity, Context-Aware Suggestions, Intelligent Coding, Performance Optimization, Real-Time Error Detection, Programming Languages, Code Analysis, Logical Consistency, Error Resolution, Developer Tools, Automated Coding Assistance.*

How to Cite: Drishya P; Sheerin Farjana M; Jagath M D; Dinesh M (2025). AI Smart Real Time Code Autocompletion, Error Fixing, Code Conversion and Optimization-Fixerbot. *International Journal of Innovative Science and Research Technology*, 10(4), 1797-1803. https://doi.org/10.38124/ijisrt/25apr695

I. INTRODUCTION

In the ever-evolving world of software development, developers are under constant pressure to write clean, efficient, and bug-free code while meeting tight deadlines. With the growing complexity of programming languages and development frameworks, managing syntax errors, logic flaws, and performance optimization becomes increasingly challenging. While traditional development tools provide some assistance, they often fail to offer the level of intelligence and automation needed to streamline the coding process and reduce manual effort. The result is a development cycle that is prone to errors, inefficiencies, and slower project completion times. To address these challenges, the AI Smart Code Autocompletion, Error Fixing, and Optimization project introduces an intelligent, AI-powered tool that significantly enhances the coding experience. This tool combines real-time autocompletion, error detection, and optimization into one seamless platform, empowering developers to write code faster, with fewer mistakes, and greater performance. The autocompletion feature predicts and suggests code snippets as the developer types, saving valuable time and reducing the risk of syntax errors. These suggestions are context-aware, ensuring they align with the developer's intentions and coding style. Beyond simple autocompletion, the tool

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr695

incorporates advanced error detection capabilities. It continuously monitors the code for bugs and inconsistencies, providing immediate suggestions for fixing errors as they arise. This feature eliminates the need for developers to spend excessive time debugging, allowing them to focus on higher-level tasks and logic development. The real-time feedback ensures that issues are identified and corrected early in the development process, leading to more stable and reliable code. Moreover, the tool offers performance optimization suggestions, analyzing the code for inefficiencies. It evaluates algorithms, data structures, and control flow, providing recommendations to improve runtime efficiency, reduce resource consumption, and enhance overall application performance. By automating these tasks, the tool helps developers write more efficient code and avoid common performance pitfalls. With this integrated approach to coding assistance, the tool transforms the development process, reducing errors, improving productivity, and ultimately delivering high-quality software solutions in less time.

II. LITERATURE SURVEY

- *Ayman Odeh, Nada Odeh, Abdul Salam Mohammed 2024 "A Comparative Review of AI Techniques for Automated Code Generation in Software Development: Advancements, Challenges, and directions"*

In software development, Artificial Intelligence (AI) is increasingly transforming the process of translating software requirements into functional code, especially during the implementation phase. This phase traditionally requires significant manual effort, but AI-powered Automated Code Generation (ACG) techniques can streamline the process, reducing both time and errors, thereby boosting productivity. This paper provides a comprehensive review of various AI techniques used for ACG, examining both traditional and advanced methods. It highlights the current state of AI in ACG and the challenges developers face, such as AI's limitations in understanding complex requirements and ensuring the correctness and efficiency of generated code. The review evaluates several AI algorithms, including machine learning, natural language processing, and deep learning. The effectiveness of these methods is assessed using criteria like accuracy, efficiency, scalability, correctness, and generalization. These metrics help measure the performance of AI in different contexts, such as generating code for large systems or handling diverse programming languages and paradigms. The paper also explores the strengths and weaknesses of these techniques, noting that while some methods excel in automating basic tasks, they struggle with more complex algorithms and context-specific logic. It emphasizes the need for further advancements to improve AI's reliability and adaptability in generating production-quality code. Finally, the paper discusses potential future directions, proposing more sophisticated AI models to handle complex systems and the development of customizable, transparent tools for better control over generated code. AI-based ACG tools are expected to revolutionize software development, making it more efficient and accessible for developers at all levels.

- *Junghyun Kim, Kyuman Lee, and Sanghyun Choi, 2020 Machine Learning-Based Code Auto-Completion Implementation for Firmware Developers*

The transition from statistical methods to machine learning-based approaches in natural language processing, aided by artificial intelligence, has influenced many fields, including software engineering. A key application of this shift is the development of deep learning-based language models to help software engineers write code more efficiently. While many research groups have attempted to create code auto-completion tools, certain challenges have necessitated in-house solutions. For instance, security-sensitive companies like Samsung Electronics may avoid commercial tools due to the risk of source code leaks. Moreover, commercial tools may not be ideal for specific domains, such as SSD firmware development, where predicting unique code patterns and styles is crucial. To overcome these challenges, this research proposes a hybrid approach that combines machine learning techniques with advanced design methods to create a tailored code auto-completion framework for firmware developers. The system aims to enhance efficiency by predicting the next code elements based on the developer's input. Sensitivity analysis of the framework showed that deterministic design, although structured, could reduce prediction accuracy by producing unexpected results. On the other hand, the probabilistic design provides a list of potential next code elements, allowing developers to select from reasonable options. This improves prediction accuracy and overall development speed. This approach seeks to provide a more effective tool for firmware developers, optimizing the code-writing process while ensuring the accuracy and relevance of predictions, particularly in the domain of SSD firmware development.

- *By Yang Qianyi, 2021 Exact Evaluation of AI-Generated Python Code: A Comparative Think around over Energetic Programming Tasks*

In afterward a long time, AI-based code colleagues have finished up compelling gadgets in program progression, updating code period and quality. Be that as it may, their ampleness shifts, and understanding their qualities and inadequacies is crucial for perfect utilize. This consider evaluates the capabilities of four driving AI code collaborators: GitHub Copilot, Microsoft Copilot, Tabnine, and ChatGPT. The explore centers on whether the AI-generated code is valuable, compelling, and practical, as well as recognizing ranges for alter. Qianyi's consider compares AI-generated code based on four key estimations: rightness, cyclomatic complexity (McCabe complexity), capability, and code assess. Rightness was measured by the rate of error-free code; McCabe complexity assessed the assistant complexity of the code; efficiency centered on execution execution; and code gauge was based on the number of lines made. The rebellious were attempted with a standard set of 100 programming prompts for solid comparison. The study's revelations allow a few bits of information. GitHub Copilot fulfilled the most hoisted rightness rate, making error-free code 42% of the time. ChatGPT made the most complex code, with a McCabe complexity score of 2.92, and additionally performed the

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr695

best in terms of efficiency. Tabnine conveyed the most brief code, while GitHub Copilot and ChatGPT made the longest. Qianyi concluded that AI-based code colleagues hold extraordinary potential for making strides code time and progression adequacy but are not in any case competent of totally supplanting human engineers, especially for complex errands. These disobedient are fruitful for less troublesome coding assignments and monotonous shapes but require help change to handle complex program cleverly. The consider prescribes that overhauling dependence organization and coordination the qualities of unmistakable devices might through and through advance the field.

- *Ayman Odeh, Nada Odeh,* **"***Exploring AI Innovations in Automated Software Source Code Generation: Progress, Hurdles, and Future Paths"*

In today's rapidly evolving software development landscape, the need for fast, efficient, and high-quality code generation has become a significant challenge. Automated Software Source Code Generation (ASSCG) has emerged as a promising solution to address this challenge, offering benefits such as speed, accuracy, and scalability. This paper examines the role of automated code generation in modern software development, focusing on the transformative potential of AI innovations and the challenges that accompany its implementation. The research highlights the critical role ASSCG plays in accelerating the development process, reducing human error, and enabling developers to focus on higher-level tasks. By leveraging AI technologies such as deep learning and evolutionary algorithms, ASSCG can enhance the efficiency and precision of code generation. These AI-driven approaches optimize code generation by learning from vast amounts of data and adapting to the evolving needs of software projects. The paper also identifies several hurdles that developers face in implementing ASSCG, including the challenge of maintaining code quality, ensuring compatibility across programming languages, and integrating AI solutions into existing development workflows. Furthermore, the study emphasizes the need for continued research and development to overcome these obstacles and unlock the full potential of AI in automated code generation. Finally, the paper explores future directions for ASSCG, suggesting that advancements in machine learning models, improved algorithms, and better integration with development tools will play a pivotal role in shaping the future of automated code generation. By addressing the challenges and leveraging AI's transformative capabilities, the field of software development can achieve unprecedented levels of efficiency and innovation.

- *Tilen Hlis, Luka Cetina, Tina Beranic, Luka Pavlic (2023) "Evaluating the Usability and Functionality of Intelligent Source Code Completion Assistants: A Comprehensive Review"*

As artificial intelligence advances, intelligent source code completion assistants have become more powerful, addressing challenges that traditional assistants struggle with. Conventional tools present suggestions in alphabetical order, requiring developers to manually search for the most relevant options. To overcome this limitation, AI-powered assistants analyze code context and extract patterns from related projects to provide more relevant suggestions. This paper presents a systematic literature review of intelligent code assistants, evaluating four prominent tools. GitHub Copilot stood out for offering complete code suggestions, marking a major advancement in assisting developers. Despite these capabilities, a survey among developers revealed mixed results. While intelligent assistants were expected to be highly useful, feedback indicated usability improvements are necessary. Interestingly, experienced developers found these tools more beneficial than less experienced ones. Additionally, the research uncovered a surprisingly low net promoter score (NPS) for intelligent assistants, indicating that their reception among developers is less favorable than expected. This suggests that while AI-driven assistants offer significant potential, their usability and adoption remain key challenges. The study emphasizes the need for further development and refinement to improve these tools' usability. Enhancing user experience, refining suggestion accuracy, and ensuring seamless integration with development workflows will be crucial in maximizing their effectiveness. In conclusion, while intelligent assistants hold promise for improving source code completion, they must become more user-friendly and efficient, particularly for novice developers. Future advancements in AI, better algorithms, and improved interfaces will be essential for increasing their adoption and effectiveness in modern software development.

- *Juan Cruz-Benito, Sanjay Vishwakarma, Francisco Martin-Fernandez, Ismael Faro (2021) "Automated Source Code Time and Auto-Completion Utilizing Significant Learning: Comparing and Looking at Current Tongue Model-Related Approaches"*

In afterward a long time, significant learning-based tongue models have picked up thought for their capacity to create human-like substance. A key application of these models is in programming, where they empower code auto-completion, period, and examining. In show disdain toward of extending captivated, there is a require of observational considers comparing particular significant learning structures for programming-related assignments. This paper analyzes diverse neural orchestrate models, tallying Typical Stochastic Point Dive (ASGD) Weight-Dropped LSTMs (AWD-LSTMs), AWD-Quasi-Recurrent Neural Frameworks (QRNNs), and Transformer models, in building lingo models for source code. Utilizing a Python dataset for code period and mask-filling errands, the makers evaluate these models with trade learning and unmistakable tokenization strategies. The consider highlights execution contrasts, qualities, and imprisonments of each plan, giving bits of information into their real-world congruity. Revelations reveal that while a few models surpass desires in specific errands, others fight with capability and accuracy in honest to goodness programming circumstances. The think almost underscores the require for development evaluation of significant learning-based tongue models for computer program planning applications. Additionally, gaps in methodology and commonsense execution challenges are recognized, proposing that no

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr695

single illustrate rules over all errands. In conclusion, the paper emphasizes the potential of significant learning for computerized code period and auto-completion but in addition highlights essential locales for upgrade. Future movements in appear adequacy, precision, and flexibility will be essential for making these systems more down to earth in real-world programming errands. Empower ask around and refinement are basic to move forward the practicality of lingo models in program headway.

- *Existing System:*

The existing system of Fixer Bot, an AI-powered Real-Time Code Autocompletion, Error Fixing, Code Conversion, and Optimization tool, is designed to streamline software development by automating various coding tasks. It assists developers by predicting code, fixing errors, converting code, and optimizing performance across multiple programming languages such as Python, Java, and C++. This automation reduces manual effort, minimizes human error, and accelerates the coding process. The Autocompletion feature analyzes real-time inputs and provides context-aware suggestions to help developers write code faster and with fewer mistakes. The Error Fixing module detects syntax, logical, and runtime errors, offering AI-driven corrections to enhance code quality. The Code Conversion component translates code seamlessly between different programming languages while maintaining logical integrity and functionality. Additionally, the Optimization module evaluates the written code, identifies inefficiencies, eliminates redundancies, and suggests performance improvements to ensure scalability and maintainability. By integrating autocompletion, error detection, conversion, and optimization, Fixer Bot provides a comprehensive AI-driven development assistant. It enhances productivity, ensures high-quality code, and supports best coding practices, empowering developers to focus on building efficient, secure, and optimized software solutions.

- *Proposed Approach:*

Fixer Bot is an AI-powered gadget sketched out to move forward the computer program enhancement plan by giving real-time code autocompletion, sagaciously botch settling, reliable code change, and beneficial optimization. Leveraging fabricated experiences, machine learning, and characteristic lingo taking care of (NLP), Fixer Bot computerizes complex coding errands, diminishes human botches, advances code quality, and animates progression workflows. One of Fixer Bot's center highlights is real-time code autocompletion, which predicts and prescribes critical code pieces as originators sort. The AI appear analyzes the setting, as of now composed code, and expand structure to offer correct and adroitly proposals, making a contrast originators compose code speedier while minimizing sentence structure botches. The Bumble Settling module recognizes and settle dialect structure, coherent, and runtime botches utilizing AI-driven examination. By leveraging significant learning and plan affirmation, Fixer Bot recognizes potential issues in the code and proposes therapeutic measures, checking best sharpens for advanced code unflinching quality. It highlights hazardous zones and gives context-aware clarifications to offer help engineers get it and resolve botches capably. The Code Alter highlight enables steady elucidation of code between various programming lingos while keeping up steady consistency and convenience. Fixer Bot utilizes an AI-based center representation approach to special code method of reasoning and layout it accurately to the sentence structure of the target lingo. By determinedly planning on unending datasets over diverse programming lingos, Fixer Bot overhauls the precision and adequacy of code alter. The Optimization module evaluates the composed code to recognize inefficient viewpoints, overabundance operations, and locales for execution improvement. By analyzing coding plans, Fixer Bot proposes elective calculations, predominant data structures, and refactoring methodologies to make strides viability, coherence, and reasonability. It ensures that the made code takes after best coding sharpens and remains versatile for future alterations. Fixer Bot additionally planning security examination by distinguishing potential vulnerabilities such as SQL mixture, cross-site scripting (XSS), and buffer surges. It gives noteworthy bits of information, endorses secure coding sharpens, and cautions engineers around potential threats to progress application security. Sketched out for reliable integration into well known IDEs, web stages, and browser extensions, Fixer Bot gives an intuitively client interface (UI) that grants engineers to get to its highlights effortlessly. It offers real-time feedback, inline suggestions, and examining offer assistance, making it an crucial AI-driven collaborator for program creators. By leveraging advanced AI and machine learning, Fixer Bot changes the way code is composed, repaired, changed over, and optimized. It streamlines the progression plan, advances code quality, diminishes examining time, and updates computer program security—empowering engineers to build viable, secure, and high-performing applications.

The AI-Based Real-Time Code Autocompletion, Bumble Settling, Code Change, and Optimization - Fixer Bot utilize case chart talks to the interaction between engineers and system functionalities. Key performing specialists consolidate originators and the Fixer Bot system, with utilize cases such as code autocompletion, botch area and settling, code alter, and optimization. The system shapes input code, analyzes botches, gives cleverly suggestions, changes over between tongues, and optimizes execution.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr695

- *Use Case Diagram:*

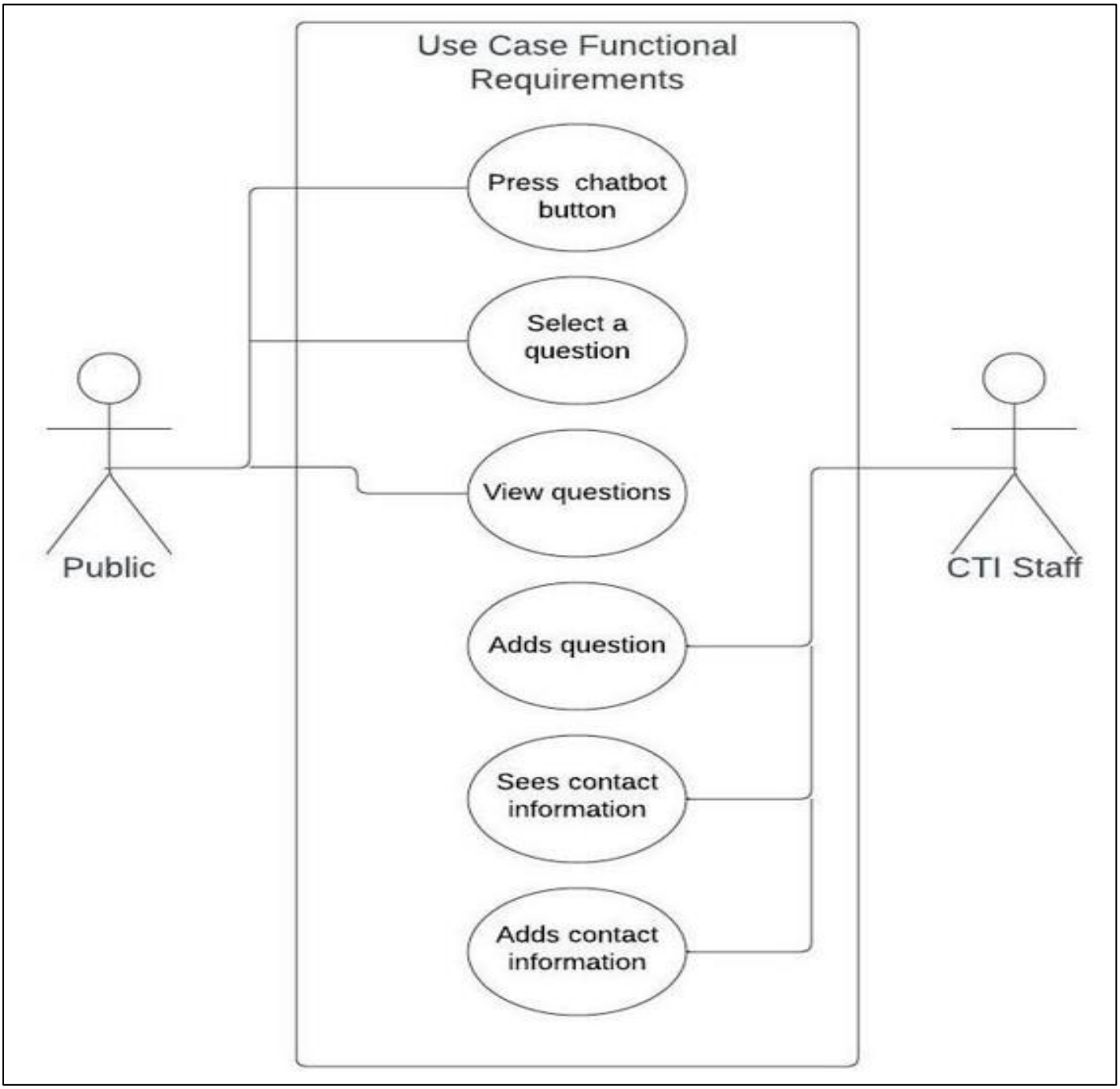

Fig 1 Use Case Diagram

➢ *List of Modules:*

- *Code Compiler*

✓ Supports multiple programming languages with real-time error detection.
✓ Provides instant feedback on compilation status and execution results.

- *Choose Language*

✓ Allows selection of preferred programming language before coding.
✓ Ensures syntax highlighting and language-specific optimizations.

- *Run:*

✓ Executes code instantly with real-time output and error detection.
✓ Displays execution time and debugging insights.

- *Translate*

✓ Converts code seamlessly between different programming languages.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr695

✓ Maintains logic and functionality while offering AI-driven improvements.

- *Debug*

✓ Identifies syntax, logical, and runtime errors.
✓ Suggests AI-powered fixes and optimizations.

- *Generate Code*

✓ Automatically generates structured and optimized code snippets.
✓ Provides AI-powered suggestions for boilerplate code and functions.

- *Console*

✓ Offers an interactive environment for executing commands.
✓ Displays runtime outputs, debugging logs, and variable states.

- *Flow Diagram:*

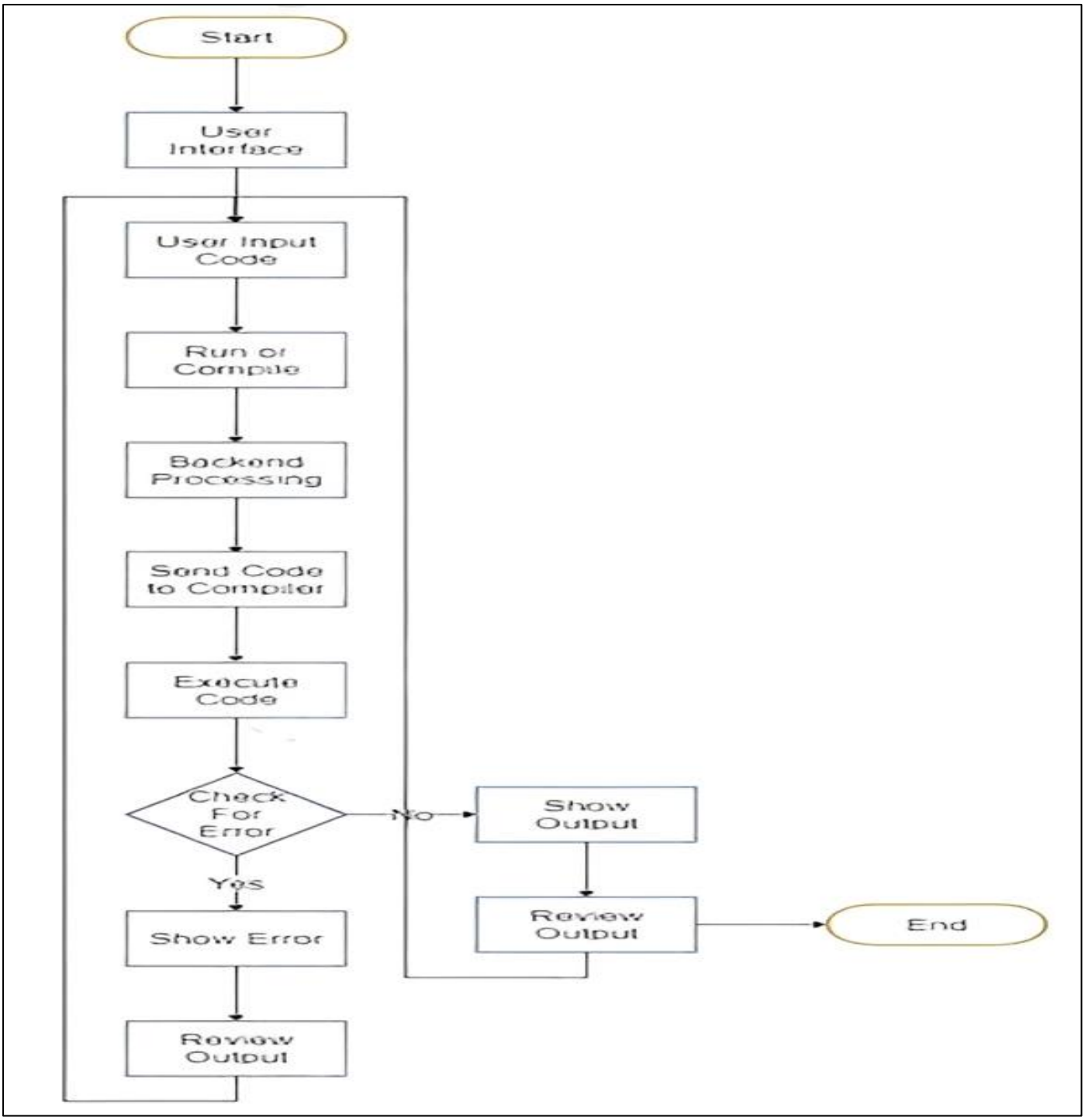

Fig 2 Flow Diagram

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr695

III. RESULT

The AI-Based Real-Time Code Autocompletion, Error Fixing, Code Conversion, and Optimization system, Fixer Bot, enhances coding efficiency by automating critical development tasks. This AI-powered tool ensures accuracy, speed, and seamless integration across multiple programming languages, including Python, Java, and C++. The Autocompletion feature predicts and suggests relevant code snippets in real time, reducing manual effort and improving productivity. Error Fixing identifies syntax, logical, and runtime errors while providing AI-driven corrections to enhance code quality. Code Conversion enables seamless translation between different programming languages, preserving logic and functionality to facilitate cross-language development. Optimization analyzes the written code, detects inefficiencies, and suggests performance improvements, ensuring scalability and maintainability. With its AI-driven analysis, Fixer Bot not only detects issues but also provides intelligent recommendations to improve structure, performance, and security. It integrates with popular IDEs, offering real-time feedback to developers. Additionally, its debugging capabilities assist in identifying and resolving runtime errors efficiently. Designed for scalability and continuous learning, Fixer Bot evolves with advancements in AI, ensuring ongoing improvements in code accuracy, security, and performance. By integrating these features, Fixer Bot transforms software development, making it faster, error-free, and optimized for modern programming needs.

IV. CONCLUSION AND FUTURE WORK

In summary, the AI-Based Real-Time Code Autocompletion, Error Fixing, Code Conversion, and Optimization - Fixer Bot enhances software development by leveraging advanced AI techniques. It streamlines coding workflows, reduces human errors, and improves efficiency through intelligent autocompletion, real-time error detection, seamless code conversion, and automated optimization. By integrating AI-driven solutions, Fixer Bot enhances accessibility, accuracy, and reliability across multiple programming languages and development environments. The incorporation of AI-based coding assistance can transform the way developers write, debug, and optimize code, ensuring faster development cycles and improved software quality. Future advancements may include enhanced deep-learning models for contextual code suggestions, adaptive learning from user preferences, and real-time collaborative coding enhancements. Additionally, improved security analysis features can further strengthen code integrity, detecting vulnerabilities and preventing security risks. By continuously optimizing these AI-driven capabilities and integrating them with evolving technologies, Fixer Bot can contribute to a more efficient, intelligent, and secure coding environment. Implementing these innovations on a larger scale will drive digital transformation in software development, making coding more intuitive, accessible, and optimized while maintaining high standards of performance, security, and scalability.

REFERENCES

[1]. C. B. Kenrad Weiss, "A Language-Independent Examination Encourage for Source Code," in Fraunhofer- AISEC, Germany, 2022.
[2]. X.X.X.Z.Y. L. a. P. D. Store Fan, "Direct Code Examination in the Al Time: An Indepth Examination of the Concept, Work, and Potential of Brilliantly Code Examination," in Unsavory crawly Collect, China, 2023.
[3]. N.FR.1.1 M. Christie Thottam, "Brilliantly Python Code Analyzer (IPCA)," Wide Journal of creative examine contemplations (UCRT), pp. 1-11, 2024.
[4]. N.M.S.SI.L.R.Z.RFA.R.T.N.N. H. W. a. H. H. Chongzhou Tooth, "Clearing Tongue Models for Code Examination: Do LLMS Really Do Their work?," around the world Journals, vol. 1, pp. 1-18, Walk 2024.
[5]. David A. Plaisted, Source-to-source graph and program building, Journal of Computer program Organizing and Applications, 2013, 6, 30-40.
[6]. P.M.N.G. Ms. Naziya Shaikh, "Advance of midway Appear up up for Source to Source Adjust," IOSR Journal of Computer Engineering
[7]. Camacho, Kim Mens, APPAREIL: A instrument for building Computerized Program Mediators Utilizing Clarified Etymological businesses, Proc. 23rd IEEE/ACM Around the world Conference on Mechanized Program Building (ASE 2008), pp. 489-490, 2008.
[8]. P. J. L. Wallis, Balanced Lingo Adjust and Its Put in the Move to Ada, Proc. 1985 each year ACM SIG Ada around the world conference on Ada (SIGAda ‘85), Vol. 5, No. 2, 1985, pp. 275-284.
[9]. David A. Plaisted, Source-to-source clarification and computer program organizing, Journal of Program Building and Applications, 2013, 6, 30-40.