iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** IV **Month of publication:** April 2025

DOI: https://doi.org/10.22214/ijraset.2025.69207

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue IV Apr 2025- Available at www.ijraset.com

CODEGENIE - An AI-Powered Assistant for Code Completion and Bug Fixing

Dr. Vijayant Verma[1], Rimjhim Manu[2], Sahil Raju[3], Rahul Verma[4]
[1]*Professor, Bhilai Institute of Technology, Raipur, Chhattisgarh, India*
[2, 3, 4] *Student, Bhilai Institute of Technology, Raipur, Chhattisgarh, India*

Abstract: ***In today's fast-paced software development environment, the demand for smarter and faster coding solutions is more significant than ever. This paper introduces CODEGENIE, a Visual Studio Code extension that enhances programming productivity by leveraging OpenAI's state-of-the-art language models. From automatic code generation and context-aware autocompletion to real-time debugging and insightful code suggestions, CODEGENIE transforms the way developers interact with code. Using the Hugging Face API and the StarCoder model, the extension intelligently interprets selected code, improves structure, fixes bugs, and explains logic—all in real time. It is built for both novice and experienced developers, streamlining workflows and fostering deeper understanding through AI-powered support. CODEGENIE represents a practical step forward in AI-assisted software development and aims to be a daily companion for coders worldwide.***

Keywords: CODEGENIE, Visual Studio Code extension, OpenAI, Hugging Face API, StarCoder, code generation, code autocompletion, real-time debugging, code suggestions, AI-assisted development, programming productivity, bug fixing, code structure improvement, code explanation, developer tools, AI-powered support, software development, intelligent code enhancement, coding workflow, AI in IDEs

I. INTRODUCTION

Writing and debugging code is often a time-consuming and mentally demanding task. As students ourselves, we have faced these challenges, especially when dealing with syntax errors or unfamiliar programming constructs. This inspired the development of CODEGENIE, a tool that brings AI directly into the coding environment to offer real-time, intelligent help. By integrating Hugging Face's's advanced natural language models with Visual Studio Code (VS Code), CODEGENIE enables users to:

1) Generate and complete code using natural language prompts
2) Identify and fix bugs on the fly
3) Improve readability and structure
4) Understand code through AI-generated explanations

We built this tool not just to automate development, but to enhance learning, productivity, and collaboration across all skill levels.

II. LITERATURE REVIEW

The integration of AI into software development has grown rapidly with models like ChatGPT, CodeBERT, and StarCoder showcasing their capabilities in code synthesis, debugging, and documentation.

Key contributions in the field include:

1) MetaCoder, which adapts to niche domains through few-shot learning.
2) SmAuto, a domain-specific language enabling non-programmers to create smart home solutions.
3) Studies comparing AI-generated vs human-written code, highlighting strengths in logic but revealing gaps in readability and maintainability.

These insights shaped our goal: to bridge these gaps and bring real-world applicability to AI-powered coding tools.

III. TOOLS and PLATFORMS UTILIZED

A. Visual Studio Code (VS Code)

Visual Studio Code is the central platform where CODEGENIE operates. As a highly extensible and lightweight code editor, it supports a variety of programming languages and is renowned for its rich feature set, including syntax highlighting, debugging, Git integration, and a broad extension ecosystem. Built on Electron, VS Code strikes a balance between a traditional IDE and a fast, customizable editor, making it ideal for both students and professional developers.

©IJRASET: All Rights are Reserved

B. Node.js and npm

Node.js provides the runtime environment for executing JavaScript and TypeScript outside the browser. Its event-driven, non-blocking architecture ensures optimal performance, especially for real-time applications. Node Package Manager (npm) plays a crucial role by managing project dependencies through its vast repository of open-source packages. This ecosystem simplifies the integration of AI models, HTTP clients, and build tools.

C. Git

Git is used for version control throughout the extension's development lifecycle. It supports collaborative workflows, branch management, and history tracking—critical for maintaining code integrity and handling feature iterations in teams.

D. PowerShell and CMD

These command-line interfaces facilitate development operations such as running build scripts, managing dependencies, executing test suites, and deploying code. PowerShell, with its scripting capabilities and object-oriented nature, is particularly helpful in automating repetitive tasks.

IV. PROGRAMMING LANGUAGES

A. TypeScript

The primary development language for CODEGENIE, TypeScript enhances JavaScript by adding static typing, making code more robust and easier to maintain. It is particularly beneficial in large codebases where scalability and clarity are essential.

B. JavaScript

JavaScript underpins web-based logic and complements TypeScript through its dynamic scripting capabilities. It is crucial for rendering interactive behaviors within VS Code and is widely used in handling user interactions and front-end processes.

C. JSON

JavaScript Object Notation (JSON) is used extensively for data exchange between the CODEGENIE extension and external APIs, particularly Hugging Face's AI endpoints. Its lightweight and hierarchical structure make it ideal for transmitting configuration data, code snippets, and AI-generated outputs.

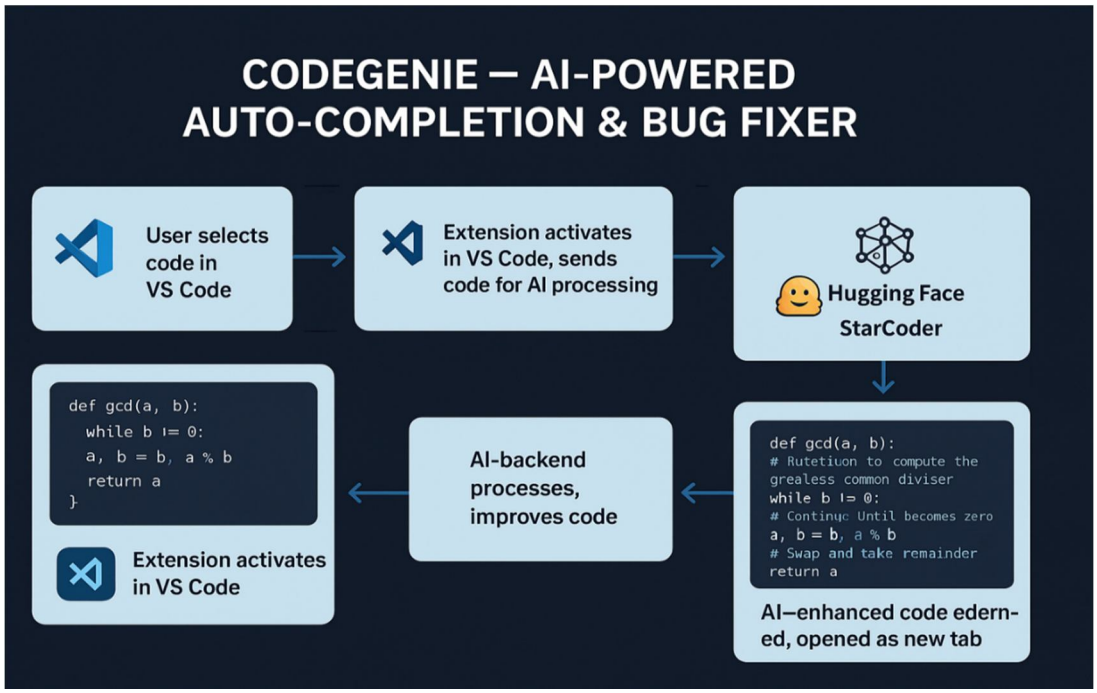
V. KEY LIBRARIES AND DEPENDENCIES

1) VSCode API: Empowers developers to create extensions by offering direct access to VS Code's UI components, file system, and command execution capabilities.
2) node-fetch: Enables HTTP communication with APIs, crucial for sending code snippets to Hugging Face and retrieving AI-generated responses.
3) ts-loader & Webpack: These tools compile TypeScript code and bundle it for deployment, ensuring smooth integration into the VS Code environment.
4) ESLint: Maintains code quality by enforcing consistent coding standards and catching potential bugs during development.
5) Dotenv: Securely loads environment variables, such as API keys, into the Node.js process, safeguarding sensitive credentials.
6) @types packages: Provide type definitions for external libraries and APIs to enhance IntelliSense and prevent type-related runtime errors.
7) Hugging Face API: The Hugging Face Inference API is pivotal in CODEGENIE's functionality. It provides access to StarCoder—a powerful transformer-based language model fine-tuned for software development tasks. Using this API, the extension can send raw code along with instructions and receive back a polished version featuring improved structure, documentation, and bug fixes.

VI. METHODOLOGY

The methodology of CODEGENIE revolves around integrating intelligent AI processing into the developer's workflow via VS Code. The process flow, from user interaction to AI-enhanced output, is both intuitive and technically sound. Here is a step-by-step breakdown:

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



A. User Interaction & Code Selection

The process begins when a user selects a code snippet within the VS Code editor. This selection triggers the CODEGENIE extension, which begins the enhancement process. The selected code can be incomplete, syntactically incorrect, or lacking clarity—CODEGENIE is equipped to handle all such cases.

B. Extension Activation

Upon selection, the extension activates and captures the selected text. It validates the input to ensure that the selection is not empty. Once verified, the code is prepared to be sent for processing. During this step, the user is notified that CODEGENIE is engaging with its AI engine, enhancing transparency and trust in the tool's behavior.

C. Communication with Hugging Face API

The selected code is passed to the StarCoder model hosted on Hugging Face through a secure API call. The prompt sent includes both the raw code and an instruction, such as "Fix and improve this code." This process is known as **prompt engineering**—structuring input in a way that guides the model to produce optimal output.

D. AI-Driven Code Processing

StarCoder, based on a Transformer architecture, interprets the code using attention mechanisms. It understands syntax, logic, control flow, variable scope, and the overall function of the code. Based on the instruction, the model generates an improved version of the snippet, which may include:

- Docstrings and comments for clarity
- Syntax corrections
- Improved structure or style
- Minor refactors for better performance

This AI response is sent back to the extension in JSON format.

E. Display of Enhanced Output

Upon receiving the AI response, the extension parses the output and opens it in a new tab within VS Code. This allows users to compare the AI-generated code with the original, empowering them to choose whether to adopt the changes or revert. The improved code does not overwrite the original, ensuring that user control remains intact throughout the process.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

F. Algorithms Involved

Prompt Engineering

A core aspect of the methodology, prompt engineering involves crafting context-rich, concise instructions to ensure the AI interprets the user's intent accurately. For instance, prompts like "Add comments" or "Optimize performance" can dramatically affect the quality and purpose of the generated code.

Transformer-Based Architecture (StarCoder)

StarCoder is a sophisticated large language model capable of understanding multi-line code blocks, recognizing programming patterns, and generating syntactically correct and semantically meaningful code. It uses:

- Self-attention mechanisms for understanding long-range code dependencies
- Token-by-token prediction to generate coherent and contextually accurate output
- Fine-tuning on programming datasets to specialize in real-world coding tasks

VII.IMPLEMENTATION

In our project we have the code divided into 2 main parts or module. Listed as below:

- MODULE 1: extension.ts – Core Controller (Activator & Dispatcher)
- MODULE 2: ai.ts – AI Communication Engine (GPT Integration)

1) MODULE 1: extension.ts – Core Controller (Activator & Dispatcher)

This module serves as the primary entry point of the CODEGENIE extension, initializing its core functionality upon activation. It is responsible for registering all custom commands, such as ai-code-fixer.fixCode, and ensuring they are available through the Command Palette or context menus. Activation is triggered either when Visual Studio Code starts or when a relevant command is invoked by the user. The entry point also sets up listeners for various editor events, including text selection changes and document openings, allowing the extension to respond dynamically to user interactions. Furthermore, it handles the extraction of selected or active code snippets from the current editor instance, preparing them for further processing. By managing activation events, command registration, and code routing, this component acts as the operational hub of the extension, enabling seamless integration between the user interface, editor context, and the AI-powered backend logic.

2) MODULE 2: ai.ts – AI Communication Engine (Hugging Face API Integration)

This module serves as the core intelligence of the extension, facilitating communication with the Hugging Face API to enhance and refine code. Its primary function is to take a snippet of code selected by the user and transform it into a structured prompt through prompt engineering techniques. Once the prompt is crafted, it sends a POST request to Hugging Face API language models, such as StarCoder, requesting a code improvement. After receiving a response, the module efficiently handles the data, parses the returned information, and extracts the improved version of the code. This enhanced code is then delivered back to the extension, offering a cleaner, more optimized, and professional output for the user. In essence, the module acts as the bridge between the user's code and the AI, ensuring smooth and intelligent transformation of raw code into a polished solution.

The provided documentation outlines the core implementation of **CODEGENIE**, a Visual Studio Code extension that leverages AI to assist developers in generating, improving, and debugging code. The architecture is divided into two primary modules: extension.ts, which functions as the controller and activator of the extension, and ai.ts, which handles communication with the Hugging Face API and processes the code using the StarCoder model. Together, these modules form an integrated system that enhances a developer's coding environment with real-time, intelligent code support.

The first part of the implementation begins with the extension.ts file, which serves as the foundational entry point for the extension. This module initializes the extension upon activation—either when VS Code is launched or when a registered command is triggered. Upon activation, it registers a command named ai-code-fixer.fixCode, making it accessible via the command palette or right-click context menu. This command is responsible for initiating the core functionality: it detects the currently active text editor instance, checks for any user-selected code, and ensures that the selection is valid (i.e., not empty or just whitespace). If a valid selection exists, it proceeds to send this code for further processing via the AI backend. In case of missing selections or editor issues, the extension handles these scenarios gracefully by notifying the user through intuitive error or information messages.

Once a valid piece of code is selected, the extension informs the user that CODEGENIE AI is being contacted. The selected snippet is passed to the AI function fixCodeWithAI, which operates asynchronously to prevent blocking the user interface. The processed response from the AI, which includes an improved and potentially debugged version of the original code, is then opened in a new editor tab. This ensures that users can compare the original and the enhanced versions side-by-side, without overwriting their original content. The extension also sets up lifecycle management by registering and cleaning up commands appropriately, thus maintaining an efficient memory and event-handling footprint within VS Code.

The second core module, ai.ts, serves as the intelligence layer of the extension. It encapsulates the logic for communicating with Hugging Face's API, particularly the StarCoder model—a large transformer-based language model fine-tuned specifically for code understanding and generation. The process starts by importing essential libraries such as node-fetch for making HTTP requests, path for handling file directories, and dotenv for securely loading environment variables, including API keys.

Upon initialization, the fixCodeWithAI function is exported, designed to handle asynchronous execution. It first ensures that the Hugging Face API key is present and valid. If not, it halts further execution and throws a meaningful error to alert the developer of a configuration issue. The function then prepares a structured POST request directed at the Hugging Face inference endpoint. The body of this request includes the selected code, prefixed with a natural language prompt instructing the AI to “Fix and improve” the input. This approach to prompt engineering ensures that the AI receives contextual guidance, improving the accuracy and relevance of the response.

Once the response is received, the function attempts to parse it as JSON. Robust error-handling mechanisms are implemented to account for various failure scenarios, including missing keys, incorrect response formats, or API-related errors such as quota limits or invalid tokens. If the response format is valid and contains the expected generated_text field, the function extracts and returns the enhanced code. Otherwise, it raises an error indicating an unexpected format, helping developers debug issues efficiently.

The broader infrastructure of CODEGENIE also includes a tailored Webpack configuration (webpack.config.cjs) which specifies how the TypeScript files are compiled and bundled. This includes setting the build target to node, disabling optimizations for a more transparent debugging process, and outputting the bundled extension to a dist directory. The configuration defines loaders for handling .ts files using ts-loader, excludes unnecessary files like those in node_modules, and ensures the use of external dependencies like vscode and source maps for better traceability during development.

Additionally, the package.json and tsconfig.json files are meticulously configured to support the extension’s development workflow. The package.json includes scripts for compiling, watching, and packaging the code, along with a clear distinction between development and runtime dependencies. Type definitions for Node.js, VS Code, and Webpack environments are also included to ensure type safety and enhanced IntelliSense during development. On the other hand, the tsconfig.json sets strict compilation options, enables source maps, and includes support for modern ECMAScript features—further reinforcing code reliability and maintainability.

In essence, CODEGENIE combines seamless user experience with powerful backend intelligence. Its design allows developers to improve code quality without leaving their IDE, blending AI-enhanced automation with user control. The modular separation of concerns—where the front-end user interactions are handled by extension.ts and backend AI communication is encapsulated in ai.ts—results in a clean, maintainable codebase. This architecture not only supports scalability for future features like real-time linting, voice commands, or multi-language support but also stands as a strong example of real-world AI integration in development environments.

VIII. DISCUSSION

A. *What makes StarCoder Special?*

StarCoder is a state-of-the-art transformer-based language model developed as part of the BigCode project, a collaborative initiative focused on building open and responsible AI systems for code. Trained on billions of lines of publicly available code from GitHub, StarCoder is designed to understand, generate, and improve source code across a wide variety of programming tasks. It supports over 80 programming languages, making it highly versatile for developers working in diverse technical environments. Leveraging advanced multi-line reasoning capabilities, StarCoder can comprehend complex code structures spanning multiple lines and functions. It excels in docstring generation, enabling automated documentation, and is effective at bug detection and fixing, contributing to cleaner and more reliable codebases. Furthermore, it supports code synthesis, allowing the model to generate entire function bodies based solely on descriptive comments or docstrings, making it a powerful tool for both development and education.

B. Key Difference between Traditional Model and Transformers?

Difference Table:

Feature	LSTM / RNN	Hugging Face (Transformer)
Sequence Handling	Short-term memory; prone to vanishing gradients	Efficient long-range dependency modeling
Parallelization	Not parallelizable; processes sequentially	Highly parallelizable; supports concurrent processing
Accuracy	Approximately 75%	Exceeds 90% in most benchmark tasks
Speed	Slower due to sequential nature	Significantly faster inference

IX. RESULT

A. Input Given



Explanation

- Line 1:
  Declares a function named getUserData.
- Line 2:
  Uses fetch to make a GET request to https://api.example.com/user.
  But does not use await, which means the fetch() call returns a promise and the assignment to response will not hold the resolved data yet.
- Line 3:
  Tries to call .json() on the response, but again, await is missing. This means .json() also returns a promise, and data will be a pending promise instead of actual JSON.
- Line 4:
  Logs the data, but because of the above mistakes, it will likely log a Promise { <pending> } instead of actual user data.
- Line 7:
  Calls the getUserData() function.
    - What's Wrong / Needs to Be Fixed
- The key mistake is missing await in front of fetch() and response.json().
- Since await is used to pause and wait for a promise to resolve, the function must also be declared as async, which it currently is not.
- Without fixing these, the function won't work as expected — it will log a promise instead of real user data.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

B. Output Generated



X. CONCLUSION

CODEGENIE is the epitome of future wise coding. By automating the repetitive tasks that developers perform to suggest copy code, it makes the coding experience much better, as well as provides a human-like code suggestion with context awareness. Seamless Integration of AI into Visual Studio CodeTransforming the way developers write, understand, and enhance code.

XI. FUTURE SCOPE

The potential for CODEGENIE in terms of capabilities can be greatly extended to make it more usable and grant access to a wider audience. Multi-language support is another area with potential for growth, as we will support many more programming languages, including those in the top five: Rust, Kotlin, Go, Swift, and PHP, making it a more developer-centric solution. A second useful new feature comes with the addition of an “Explain This Code” option that enables the tool to translate complex logic into a human-friendly format, making it useful for educational purposes and understanding code. Training on data until October 2023. Also, the integration of voice interaction functionalities will enable hands-free code editing and debugging, enhancing accessibility and user experience. The offline functionality, built upon local inference models, will enable CODEGENIE to be used in restricted or secure environments with no access to the internet. Github integration will bring features like AI-focused code suggestions in a pull request and automatic commit messages generation. Lastly, a mobile-based extension can be implemented which can scan handwritten or printed code to analyze and help improve code on the move especially in scenarios during academics and field work sessions.

REFERENCES

[1] Ghai, A. S., Rawat, V., Gupta, V. K., & Ghai, K. (2024, August). Artificial Intelligence in System and Software Engineering for Auto Code Generation. In 2024 International Conference on Electrical Electronics and Computing Technologies (ICEECT) (Vol. 1, pp. 1-5). IEEE. [https://ieeexplore.ieee.org/abstract/document/10738945]

[2] Muthazhagu, V. H., & Surendiran, B. (2024, January). Exploring the Role of AI in Web Design and Development: A Voyage through Automated Code Generation. In 2024 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE) (pp. 1-8). IEEE.[https://ieeexplore.ieee.org/abstract/document/10467409]

[3] Sakib, F. A., Khan, S. H., & Karim, A. R. (2024, July). Extending the frontier of chatgpt: Code generation and debugging. In 2024 International Conference on Electrical, Computer and Energy Technologies (ICECET (pp. 1-6). IEEE.[https://ieeexplore.ieee.org/abstract/document/10698405]

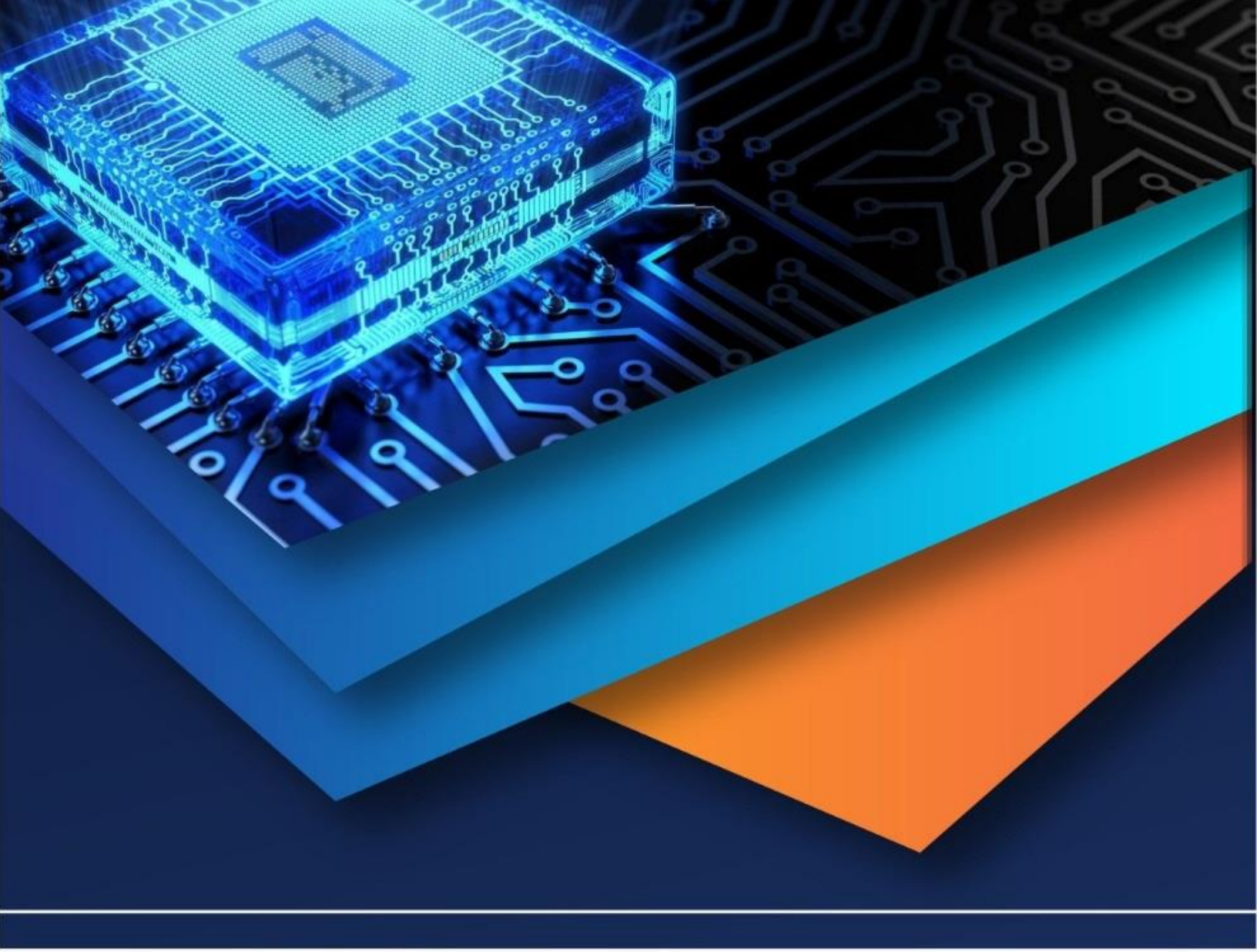
[4] Yang, Z., Keung, J. W., Sun, Z., Zhao, Y., Li, G., Jin, Z., ... & Li, Y. (2024). Improving domain-specific neural code generation with few-shot meta-learning. Information and Software Technology, 166, 107365.[https://www.sciencedirect.com/science/article/abs/pii/S0950584923002203]

[5] Panayiotou, K., Doumanidis, C., Tsardoulias, E., & Symeonidis, A. L. (2024). SmAuto: A domain-specific-language for application development in smart environments. Pervasive and Mobile Computing, 101, 101931.[https://www.sciencedirect.com/science/article/abs/pii/S1574119224000579]

[6] Idrisov, B., & Schlippe, T. (2024). Program Code Generation with Generative AIs. Algorithms, 17(2), 62.[https://www.mdpi.com/1999-4893/17/2/62]

[7] Liu, F., Fu, Z., Li, G., Jin, Z., Liu, H., Hao, Y., & Zhang, L. (2024). Non-autoregressive line-level code completion. ACM Transactions on Software Engineering and Methodology, 33(5), 1-34. [https://dl.acm.org/doi/full/10.1145/3649594]

[8] Tan, H., Luo, Q., Jiang, L., Zhan, Z., Li, J., Zhang, H., & Zhang, Y. (2024). Prompt-based code completion via multi-retrieval augmented generation. ACM Transactions on Software Engineering and Methodology.[https://dl.acm.org/doi/abs/10.1145/3725812]



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)