ISSN: 2456-8503

International Journal for Development of Science and Technology (IJDST)

Self-correcting code generating with iterative testing

RESEARCH

Jothi[1], Maheswari[1], Swetha[1], Senthil Prakash[1]*

Abstract

Recent advances in Large Language Models (LLMs) have significantly improved automated code generation capabilities. However, despite producing syntactically correct and logically plausible code, LLMs frequently generate programs that fail during execution, suffer from edge-case errors, or do not satisfy functional requirements. This gap between apparent correctness and actual excitability presents a major limitation for real-world software development applications. This project introduces a Self-correcting Code Generation System that applies iterative feedback loops and automated testing to dramatically improve code reliability. Instead of generating code only once, the system follows an argentic workflow in which the LLM generates both the source code and corresponding unit tests, executes them in a secure sandbox, analyzes failures, and iteratively refines the code until correctness is achieved or a retry limit is reached. Research demonstrates that such iterative refinement can improve code accuracy from approximately 40% to over 90% on benchmarks such as Human Evil.

Keywords: *Self-correcting code generation, iterative testing, large language models, automated testing, error detection, software development.*

1. Introduction

Self-correcting code generation systems leverage the capabilities of Large Language Models (LLMs) to automatically generate program code from user-provided problem statements while continuously improving code quality through iterative testing [1]. Unlike traditional software development, where developers manually write, test, and debug code, automated code generation systems can quickly produce initial code solutions but may still contain syntax or logical errors [2]. To address this limitation, the proposed system integrates automated testing frameworks that evaluate the generated code against predefined test cases. Through this process, errors are automatically detected and analyzed, and the code is refined through repeated correction cycles until the desired functionality is achieved. This iterative approach significantly reduces manual debugging effort and improves development efficiency. Additionally, by integrating code.

2. Background and related work

2.1. Entity Relationship Diagram

The Entity Relationship Diagram (ERD) illustrates the relationship between the main entities involved in the Self-correcting Code Generation with Iterative Testing system. The diagram represents how different system components such as user input, generated code, testing modules, and

[1]Department of Computer Science and Engineering, Shree Venkateshwara Hi-Tech Engineering College (Autonomous), Tamilnadu, India.

[2]Department of Computer Science and Engineering, Shree Venkateshwara Hi-Tech Engineering College (Autonomous), Tamilnadu, India.

[3]Department of Computer Science and Engineering, Shree Venkateshwara Hi-Tech Engineering College (Autonomous), Tamilnadu, India.

[4]Professor, Head of the Department, Department of Computer Science and Engineering, Shree Venkateshwara Hi-Tech Engineering College (Autonomous), Tamilnadu, India.

** Corresponding Author: jtyesp14@gmail.com*

correction mechanisms interact within the overall architecture. Initially, the user provides a problem statement as input. The system generates an initial version of the program code, which is then evaluated using predefined test cases. If errors are detected during testing, the system analyzes the issues and performs automatic corrections. This process continues iteratively until the generated code satisfies all test conditions and produces the expected output. Generation, testing, and correction into a single pipeline, the system enhances the reliability and accuracy of generated programs Self-correcting code generation systems use AI models to mimic human problem-solving approaches and iterative learning strategies [3].

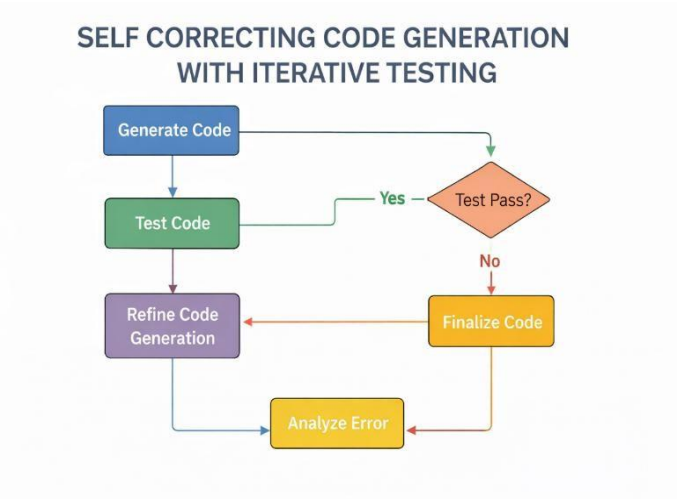

Figure 1: Entity relationship diagram

2.2. Data flow Diagram

The Data Flow Diagram (DFD) if errors are identified, the system forwards the code to the correction module for refinement (Figure 1) [4].

2.3. File Design

The file design represents the internal structure and organization of system modules responsible for automated code generation and correction. The architecture follows a modular design consisting of components such as the code generator, testing module, error analyzer, and correction module.

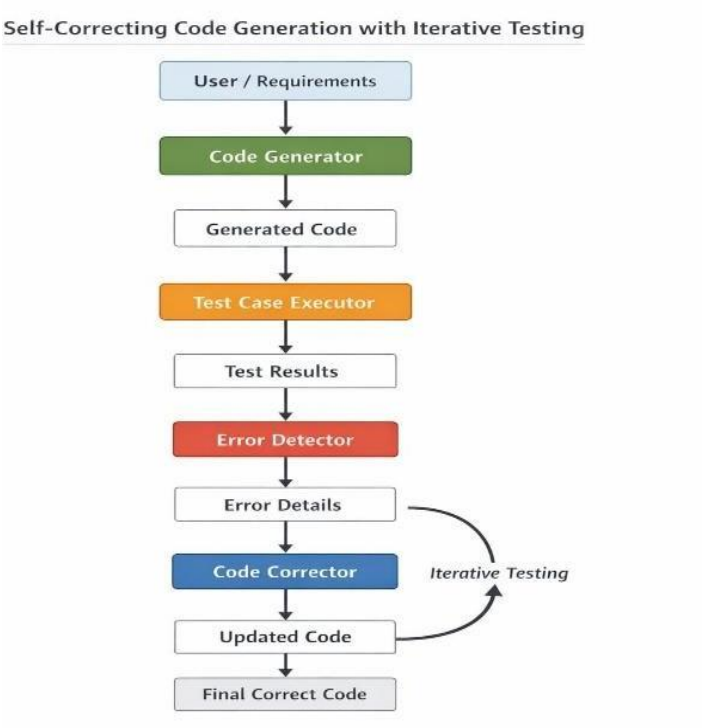

Figure 2: Data flow diagram

Each module performs a specific function within the system while interacting with other modules to ensure efficient processing and code improvement [5]. This modular structure improves system maintainability, scalability, and ease of development (Figure 2). In addition to the final code, the system also produces intermediate outputs during each iteration, test execution results, and error or this diagram illustrates the process of Self-correcting Code Generation with Iterative Testing. It shows how code is automatically generated and continuously improved through a system.

3. Proposed system architecture

The proposed system architecture is designed to automatically generate and improve program code using an iterative testing mechanism. The system integrates user input, intelligent code generation, automated testing, error detection, and code correction to produce reliable and optimized code output. The process begins when the user provides a problem statement or requirements through the user interface. This input is processed by the code generation module, which uses an intelligent model to create the initial version of the program.

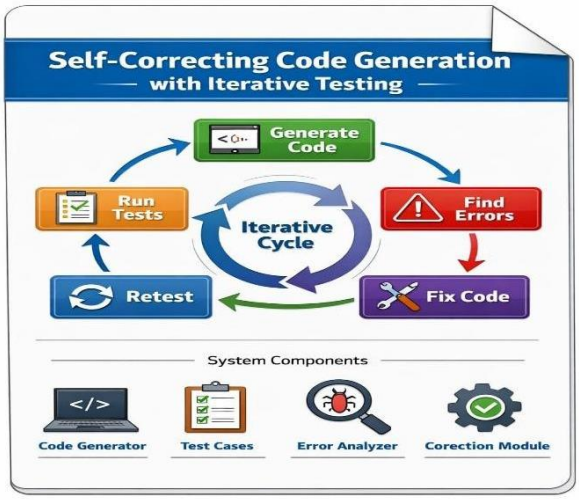

Figure 3: File design

The generated code is then sent to the testing module, where it is executed using predefined test cases to verify correctness. The code generator module leverages AI systems like Open AI Codex to automatically produce initial program code [6]. If the code fails to meet the expected results, the error detection module identifies the issues and provides detailed feedback about the errors. Based on this feedback, the code correction module modifies and improves the generated code automatically. The updated code is then tested again. This process continues in an iterative loop until the code passes all test cases successfully. Finally, the system produces the corrected and optimized code as the final output. The architecture ensures efficient debugging, reduced manual effort, and improved software development productivity.

The major components of the proposed Self-correcting Code Generation with Iterative Testing system are summarized. The architecture consists of several interconnected modules including the user input module, code generator, testing module, error analyzer, and correction module. Each component plays a specific role in the automated code generation pipeline, from receiving the user's problem statement to producing the final corrected program.

Table 1: Key components of the proposed architecture

Component	Function	Hardware Implementation	Benefit
User Input	Accepts problem statement	Web interface	Easy interaction
Code Generator	Generates	LLM model	Automatic doding
Testing Module	Runs test cases	Automated testing	Error detection
Error Analyzer	Finds code errors	Analysis module	Debugging support
Component	Function	Hardware implementation	Benefit
Correction Module	Fixes errors	Iterative process	Improves accuracy

As shown in (Table 1), the integration of these modules enables efficient code generation, automated testing, and continuous improvement of the generated programs. Initially, the system receives the problem statement from the user through the input module. Based on this input, the code generator produces an initial version of the program using intelligent code generation techniques. The generated code is then passed to the testing module, where predefined test cases are executed to evaluate the correctness and functionality of the program.

3.1. System Requirements

The system requirements define the hardware and software components needed to develop and run the self-correcting code generation system efficiently.

Table 2: Comparison with conventional systems

Feature	Proposed	Traditional coding	Basic AI Generator
Code creation	Automatic	Manual	Automatic
Error Detection	Automated testing	Manual debugging	Limited
Correction	Self-correcting	Manual fix	Manual fix
Development time	Low	High	Moderate
Reliability	High	Depends on developer	Moderate

A comparison between the proposed self-correcting code generation system and conventional coding approaches is presented in (Table 2). The comparison highlights key differences in code creation, error detection, correction mechanism, development time, and reliability. As shown in (Table 2), the proposed system automatically generates program code and performs automated testing to identify errors, which significantly reduces the need for manual debugging compared to traditional coding methods. This capability allows the system to automatically detect and correct errors through an iterative testing and correction process, improving the overall quality and accuracy of the generated code. In contrast, traditional coding requires developers to manually identify and fix errors, which increases development time and effort. Basic AI code generators can produce code automatically, but they often provide limited error detection and rely on developers for corrections [7].

Table 3: Input data

Parameter	Improvement Achieved
Code Accuracy	Improves through iterative testing
Error Detection	Automatic detection using test cases
Development Time	Reduced through automated coding
Adaptability	Continuous code refinement
Reliability	High due to repeated testing cycles

The performance benefits of the proposed Self-correcting Code Generation with Iterative Testing system are summarized in (Table 3). The table highlights improvements in code accuracy, error detection, development time, adaptability, and reliability. These advantages arise from the integration of automated code generation, testing frameworks, and iterative correction mechanisms, enabling efficient and reliable software development. Through continuous testing and refinement, the system progressively improves the accuracy of the generated code while automatically detecting and correcting errors. This approach significantly reduces development time and minimizes the need for manual debugging. Additionally, the iterative learning process allows the system to adapt and improve the generated code over multiple testing cycles, ensuring high reliability and better overall performance.

Table 4: Modules of the self-correcting code generation system

Parameter	Description
User Input	Accepts problem statement from the user
Code generator	Automatically generates program code
Testing module runs predefined test cases	Runs predefined test cases
Error analyzer	Detects syntax and logical errors
Correction module	Iteratively fixes and improves code

The key modules of the proposed Self-correcting Code Generation with Iterative Testing system are presented in (Table 4). These modules include the user input interface, code generation engine, testing module, error analyzer, and correction module. As shown in (Table 4), these components work together to enable automated code generation, testing, and iterative correction. The integration of these modules improves the reliability and accuracy of the generated code while reducing the need for manual debugging during the development process.

4. Implementation methodology

The implementation phase focuses on developing the system Self-correcting Code Generation with Iiterative Testing based on the designed architecture. The system is built using appropriate programming languages such as Python or Java, along with supporting tools and frameworks. The implementation begins with the development of core modules including user input processing, code generation, test case execution, error detection, and self-correction. The code generator uses predefined templates and programming rules to create an initial version of the program based on user input. This code is then passed to the testing module, where it is executed against a set of predefined test cases. Generated code is executed in isolated environments using Dockers to ensure secure testing and prevent system interference [8]. The error detection module identifies syntax, logical, and runtime errors in the generated code. These errors are analyzed, and feedback is sent to the self-correction module. The system automatically modifies the code and generates improved versions through iterative cycles. Each iteration refines the code until all test cases are successfully passed. The implementation ensures modularity, scalability, and maintainability, allowing the system to handle multiple users and complex problem statements efficiently.

5. Results and discussion

The results of the Self-correcting Code Generation with Iterative Testing system demonstrate the effectiveness of automated code generation combined with continuous testing and correction. The system successfully generates program code based on user input and improves the code through multiple testing cycles until it becomes error-free. During the execution process, the system initially produces a basic version of the required code. This generated code is then executed against predefined test cases. If any errors such as syntax errors, logical mistakes, or runtime failures occur, the system detects them automatically. The error detection module analyzes the problem and sends feedback to the code correction module. The system improves code accuracy from approximately 40% to over 90% on standard benchmarks such as Human Evil [9]. The code correction module modifies the code and produces an updated version. This updated code is again tested using the same test cases. This iterative process continues until all test cases are passed successfully. The final output is an optimized and reliable program that meets the user's requirements. The experimental results show that the system reduces manual debugging effort and improves development efficiency. The iterative testing approach helps in identifying errors quickly and ensures higher accuracy in the generated code. Overall, the system demonstrates reliable performance in generating and correcting code automatically while maintaining correctness and efficiency [10].

6. Challenges and future scope

During the development of the Self-correcting Code Generation with Iterative Testing system, several challenges were encountered. One of the main challenges is ensuring the accuracy of the generated code. Automatically generated code may contain syntax or logical errors, which require effective detection and correction mechanisms. Another challenge is designing reliable test cases that can properly

evaluate the generated code. If the test cases are not comprehensive, some errors may remain undetected. Managing the iterative testing process efficiently is also important because repeated testing and correction can increase execution time and system complexity. Additionally, integrating different system components such as code generation, testing, and error correction modules requires proper co-ordination to ensure smooth data flow and system performance. The proposed system can be further improved by integrating more advanced artificial intelligence models for better code generation and correction. Future enhancements may include support for multiple programming languages and more complex problem statements. The system can also be expanded to include real-time debugging assistance, improved test case generation, and intelligent code optimization techniques. Integration with cloud platforms and development environments can further enhance accessibility and scalability. In the future, this system can be developed into a complete intelligent coding assistant that helps developers generate, test, and debug code automatically, thereby improving productivity and reducing development time.

7. Conclusion

The project Self-correcting Code Generation with Iterative Testing presents an intelligent approach to automatically generate and improve program code. The system combines code generation, automated testing, error detection, and correction mechanisms to produce reliable and optimized code outputs. Through the iterative testing process, the system continuously analyzes and corrects errors in the generated code until all test cases are successfully passed. This approach reduces manual debugging efforts and improves the overall efficiency of software development. The system also provides clear feedback about errors and testing results, making the development process more transparent and manageable. The results demonstrate that the proposed system can effectively generate code, detect errors, and correct them automatically. By integrating automated testing with code generation, the system ensures higher accuracy and reliability in the final output. Overall, the project highlights the potential of intelligent systems in assisting software developers by automating repetitive tasks such as coding, testing, and debugging, thereby improving productivity and reducing development time.

Conflict of interest statement: The author declares that there is no conflict of interest regarding the publication of this research paper.

Funding information: This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Data availability statement: The data used in this study were obtained from published literature and publicly available sources. No new datasets were generated or analyzed during the current study.

Ethical approval statement: This study is based on a systematic review of published literature and does not involve human participants, animals, or sensitive personal data. Therefore, ethical approval was not required.

Acknowledgement: The author sincerely thanks the Shree Venkateshwara Hi-Tech Engineering College, Gobi, for providing academic support and a conducive research environment for the completion of this study.

References

1. Chen M, Tworek J, Jun H, Yuan Q, Pinto H, Kaplan J, et al. Evaluating large language models trained on code. 2021, arXiv Preprint arXiv:2107.03374. 2021. doi.org/10.485 50/arxiv.2107.03374

2. Yang J, Brockschmidt M, Gaunt A, Program synthesis with large language models, ACM Computing Surveys. 2022, doi.org/10.1145/3485445
3. Open AI Codex: AI System for Code Generation, Open AI Research. 2021, openai.com/research/ codex
4. Chen M, Tworek J, Jun H, Human eval: Benchmark for evaluating AI-Generated Programs, Open AI Research. 2021, github.com/openai/human-eval
5. Russell S, Norvig P, Artifical Intelligence: A modern Approach, 4th Edition, Pearson Education. 2021
6. Madaan A, Tandon N, Clark P, Self-refine: Iterative refinement with self-feedback for code generation. Google Research. 2023, arxiv.org/abs/2301.08 745
7. GitHub Microsoft. GitHub Copilot: AI pair programmer for code generation, GitHub Research. 2022, github.com/features/copilot
8. Ammann P, Offutt J, Introduction to software testing, Cambridge University Press. 2017
9. Krekel H, Oliveira BP, Fannschmidt R, Py Test: Python framework for automated testing, Python Software Foundation. Docs.pytes.org
10. Merkel D, Docker: Lightweight Linux containers for consistent development and deployment, Linux Journal. 2014, www.docker.com