Automated Software Engineer: Harnessing AI to Revolutionize Code Generation and Software Development

K. Karthik, P. Mohan, A. Mokshith, S. Ariff and G. Hemanth Kumar Yadav
Computer Science and Engineering (AI & ML), Srinivasa Ramanujan Institute of Technology, Anantapur-515731, Andhra Pradesh, India

Keywords: Natural Language Processing, Software Engineering, AI-Powered Development, Automated Coding, Intelligent Software System.

Abstract: The Automated Software Engineer project designs an AI system to accept high-level human instructions which subsequently creates code while helping software development through reduced human involvement. The goal involves the automation of various aspects throughout the software engineering process. The project implements large language models (LLMs) which include Claude 3, GPT-4, and Gemini together with complex planning and reasoning algorithms. The system implements technologies which let it consume natural language instructions while producing code and connecting with development tools.Early outcomes suggest the software engineer automation can optimize code creation together with maintenance activities and scheduling tasks. The Automated Software Engineer demonstrates capabilities to create high-quality code and shorten development cycles and enhance software quality which makes it a transformative force for code generation and software development. AI integration within the system enables transformation of the software development industry by enhancing both productivity and efficiency and innovation. The project's achievements deliver advantages to software developers as well as project managers together with organizations.

1 INTRODUCTION

AI has moved so quickly; it is now setting new frontiers for technology and software development is one of its key developments. Software makes up the backbone of every major technological progression of our modern day from healthcare to finance, education to entertainment. Traditional processes of software development require considerable time that developers spend in performing repetitive tasks like code writing and compiling for optimization. These challenges become harder and harder to deal with thanks to the growing complexity of software systems, yet there is a lack of qualified developers on the market. Software delivery requires an urgent upgrade because developers want new approaches to ensure less development workflow time and errors to deliver the software products faster. AI workloads started being executed by new, revolutionary hardware in the research that led to the second wave of AI research to cure the inefficiencies of software development.

Development tools and methodologies have made great advances to date, but there are still gaps preventing the automation of complex coding components and the reduction of human error, as well as improving development team collaboration. Present-day advancement devices center around individual abilities like static investigation and code finish yet they don't bolster the full-life cycle of programming advancement which includes manual work application for bug searching for and testing other than code transformation The manual forms bringing about postponing venture results because of strange outcomes swaying well repute and nature of programming items. October 20, 2023, 9:00 AMEastern Daylight Time Due to this advancing software system complexities requiring modern solutions with more advanced capabilities to effectively manage large source code collections while ensuring compliance to best practice guidelines. A comprehensive AI system will enhance human capabilities to perform intricate manual processes, thus rendering the current method obsolete. AI solutions that target these shortcomings

Karthik, K., Mohan, P., Mokshith, A., Ariff, S. and Yadav, G. H. K.
Automated Software Engineer: Harnessing AI to Revolutionize Code Generation and Software Development.
DOI: 10.5220/0013884600004919

In *Proceedings of the 1st International Conference on Research and Development in Information, Communication, and Computing Technologies (ICRDICCT'25 2025) - Volume 2*, pages 449-456
ISBN: 978-989-758-777-1

will enhance developer productivity and generate savings, allowing staff to take on higher order responsibilities and find opportunities to innovate in their work.

This study focused on the development of an artificial intelligence (AI) mechanism that manages the software development operations by providing automated solutions in the domains of code base function generation, error identification, and application execution efficiency. It primarily addresses the creation of AI tools that can take text inputs and create highquality code, and also help suggest changes along with scanning and fixing for vulnerabilities and defects in real time. The research also aims to foster collaborative development by suggesting developers' recommendations on demand for optimizing code while promoting standard programming practice. Using machine power to discover human creativity helps developers build solid scalable and creative Software solutions faster and more efficiently. Through its application, it applies AI at every stage of the software development process from coding to deployment all to maximise productivity and reduce manual tasks while maintaining high-quality code. The study direction is a group of AI models, one model is responsible for decoding the natural language requirements, while the other model is responsible for code analysis and generates a standard optimized solution. The research is of critical importance for several domains of business. These tools help build the product quickly and allow features to be deployed more efficiently; thus, improving the product delivery speed, so, as a result, the validation is carried out instantly with the use of such AI tools for development. With programmatic dependence, the application of AI in the DevOps pipeline permits the pipelines to be automated which brings about unending delivery and lessens mistakes. Over these tools, the specialists in AI make their efforts more beneficial by erecting complex algorithms and analyzing big datasets to propel their investigative processes which paves the way for the variety of the research atmosphere. Even though they have restricted technical resources, technical startups use AI-powered tools to create software applications with complex functionality. The findings allow for access to the best development tools, enhancing competition in the market and expanding innovation worldwide. The introduction of AI into software development practices is a entire new way of working that has huge implications for the discipline. The research research addresses major development hurdles through AI-based technologies which carry out automation operations in repetitive jobs and enhanced code structuring and teamwork capabilities. By focusing on hypothesizing, developers can focus on innovative solutions for complex issues which further accelerates software production speed and enhances output quality. These outcomes have the power to revolutionize the way individuals create software and build incredible results in startup dimension firms all the method to huge enterprises. The new era of technological transformation will be ignited by increased software development when AI will take its software development class on a larger scale. With intelligence, it is possible to develop an "Automated Software Engineer" that changes software design, production and distribution in real-time.

2 RELATED WORK

Research and development in Artificial Intelligence (AI) for software development has intensified during the previous years through multiple investigations and tool development targeting the automation and optimization of different phases of the software lifecycle. This document evaluates research conducted around objectives focusing on AI-powered code creation and error discovery as well as system performance enhancement and team-based development.

2.1 AI-Driven Code Generation

AI code generation is one of the most prominent areas of research for AI, and indeed GitHub's Copilot and OpenAI's Codex are groundbreaking tools in this space. The tools work via large language models that ingest huge code repositories and spit out blocks of code, as well as functions and even full programs, when given instructions in natural language. The study affirms these tools speed up development because they process instructions to code that is run repeatedly while lightening developer workload. Most of them involve the process to ensure that the optimal functioning of code while maintaining a safe and secure execution along with compatibility with particular project requirements. Chen et al. show that coding is well suited to transformer-based models (2021) but need better context analysis and customization options.

2.2 Error Detection and Bug Fixing

AI technology demonstrates effective capabilities for automated detection of errors while simultaneously performing bug correction tasks. The machine learning capabilities of tools such as DeepCode together with Snyk analyze code bases in order to detect vulnerabilities and potential performance issues as well as bugs. Allamanis et al. (2018) conducted research into neural networks application for static code analysis which demonstrates strong capabilities at uncovering bugs and security vulnerabilities. Several research studies by Le Goues et al. (2019) alongside other authors prove that AI-based program automation enables automated detection and application of standard bug repairs. Automated error solutions face ongoing difficulties when dealing with specific bugs requiring contextual handling and when verifying the reliability of their generated solutions.

2.3 Performance Optimization

Modern performance optimization tools using AI-based analysis help developers systematize code refactoring of programs. Grech et al. (2020) use reinforcement learning methods to study how optimization of algorithms affects computational processing times in their research article. Program maintenance and performance gains result from code optimization conducted by Facebook's Aroma tool and Google's ML-based solution. The operation of code optimization tools remains limited for big-scale real-world coding tasks and needs extensive human help for their functionality. Existing conditions necessitate the creation of flexible AI solutions that will resolve this issue.

2.4 Collaborative Development and Real-Time Assistance

Artificial Intelligence delivers better team development cooperation by applying its functions. Tabnine and Kite deliver real-time coding completion features that help teams enhance their productivity and enforce developers to follow established programming methods. The study performed by Murphy-Hill et al. (2019) explores AI support for team programming through interactive coding help revealing dynamic feedback alongside code context needs. The available tools today exhibit restricted capabilities to provide extensive end-to-end assistance thus affecting their ability to support big distributed development teams effectively.

2.5 AI in DevOps and Continuous Integration

The DevOps community shows strong dedication to AI deployment as a result of Jenkins AI and Harness tools which automatically perform build and testing pipeline operations. Artificial intelligence demonstrates its capability to optimize CI/CD operations through Humble et al. (2021) research which decreases human errors along with manual tasks. The deployment success of AI systems encounters multiple system operations obstacles and must show proven reliability during actual operational use.

2.6 AI for Startups and Resource-Constrained Teams

AI technology delivers its greatest development advantages through audio tools to teams working in small departments and startup businesses. According to Johnson et al. (2020) artificial intelligence technology provides innovative development tools for small development groups allowing them to match up against established organizations. Users of all levels benefit from AIpowered coding tools like Ghostwriter by Replit and CodeWhisperer by Amazon who access lowcost solutions to build improved code that delivers higher developer capabilities.

2.7 Challenges and Future Directions

Developers need to combine dependent code production with security compliance while enhancing their AI understanding abilities and incorporating AI tools into existing workflow systems for successful AI implementation in software development. According to Sutton et al. (2022) artificial intelligence systems require human specialist knowledge for optimal functioning outcomes. While executing AI implementations in their development framework software developers face the moral duty to address both AI bias problems and employment disruptions caused by AI.

3 EXISTING SYSTEM

Software development today depends mainly on human labor through manual processes which require developers to code and execute testing while fixing bugs. The field of development tools together with

methods has made great progress but the majority of present systems demand extensive human engagement to function. IDEs and version controls with Git and CI/CD pipelines have made operations more efficient but remain limited to independent functionalities without enabling all-encompassing automation. IDEs provide developers with code development features such as syntax highlighting together with code completion and debugging capabilities while these tools do not have the ability to code autonomously or solve complex problems independently. The automated testing and deployment processes of CI/CD pipelines rely on developers to preserve and create the program code base. The available programming tools help developers but they fail to solve the fundamental problems of total automation for software development.

New technology based on artificial intelligence offers limited automation capabilities to various areas within the software development field. GitHub Copilot with Codex from OpenAI supports developers through its large language modeling (LLM) technology that provides code snippets and line completion and develops small functions according to natural language commands. Research indicates that these programming aids both decrease developer mental strain and enhance writing speed. The current applications have restricted capabilities because they are unable to fulfill sophisticated duties including project demand interpretation along with task decomposition and automated code debugging and optimization processes. These tools fail to understand the context of the development work which results in substandard solutions or errors. Although these tools supply important help for developers their capabilities stop short of offering complete automation of software programming work.

The current version of AI tools used in software development operates in standalone platforms instead of merging these functionalities into a connected system. Each tool specializes in its domain but lacks integrated functionality with other development stages causing developers to carry out manual linkages across the workflow. A fragmented approach reduces possible efficiency potential that would come from integrated procedures. A unified system which combines AI capabilities to complete the entire software development cycle starting from code generation through debugging to project management has still not become part of the existing framework. The Automated Software Engineer project targets the development process gaps through its comprehensive solution that covers all stages of intelligent software engineering.

4 PROPOSED SYSTEM

The Automated Software Engineer project builds a system which uses large language models (LLMs) to develop a new approach for software coding automation. The system operates independently to decode human command language then creates usable code while developers need monitor it only occasionally. The model performs two successive phases starting with the plan breakdown during the planning stage followed by code generation in the implementation stage. By implementing this approach developers experience reduced cognitive workload and their generated code becomes more effective and higher quality. The system delivers autonomous information collection via planning algorithms and web browsing abilities to refine its outputs and handle complex problem-solving tasks.

When compared with direct code generation and Chain-of-Thought (CoT) methods, the devised system improves upon Pass@1 successes by 25.4% and achieves an additional 11.9% improvement over CoT systems. The result is more correct, robust, and readable code which can be validated using personal heuristics. This allows system to run in the any programming language and any IDE because of its modular and extensible architecture which helps in extensions of API and external tools. The Automated Software Engineer aims to equip developers with the tools to generate impressive, efficient software solutions that are easily maintainable while streamlining the development process and minimising the likelihood of errors.

5 METHODOLOGY

In this paper, we introduce CoderGen, an agent-based framework for generating codes on tasks in AICoderEval. This framework can construct domain-specific tasks benchmark, for training and evaluation, and then fine-tunes a code generation model on the benchmark.

5.1 Error Traceback and Analysis

CoderGen framework has the ability to catch and refine errors that may occur during the code generates process as shown in Figure 1. All this

happens during the creation phase of the framework; during these during the framework code execution occurs in a test environment. This is why an error traceback is generated when incorrect code execution take place, the code trace provides you with the entire steps of code that led to the failure. This traceback is analyzed by the framework to identify what kind of failure was the cause, was it a failure a syntax error, is it the type error or a failure in integration with either code library or an API solution?

Figure 1: Error Handling and Sentiment Analysis Debug Workflow.

CoderGen uses its fine-tuned language model to evaluate error messages before it generates possible solutions for developers through its error analysis mechanism. The framework generates suggestions by applying both an understanding of the intended code functions and the error context from the overall code system. The system presents possible fixes to users who retain the option to both implement modifications or let the system implement changes automatically for evaluation purposes. The development cycle repeats error discovery together with code evaluation followed by repairs until the program completes every tested case with the needed specifications.

5.2 Iterative Code Re-Generation

The framework advances to code regeneration mode after both errors and possible improvements have been discovered. At this phase the framework employs feedback generated from error analysis to improve its code generation approach. The package containing erroneous code along with suggestions and original instructions enters the language model for the generation of a new updated code snippet.

The updated code snippet moves into testing while the entire process of error discovery and diagnosis and code repair repeats itself. The framework implements an iterative process that produces code improvements from iteration to iteration while solving present problems. Rather than traditional methods CoderGen benefits from its framework learning power to modify code generation through real-time feedback analysis.

Series of iterative feedback cycles enables CoderGen to develop code with a high degree of correctness as well as maintainability for efficient implementation of best practices consistent with target domains. Applications of this approach lead to substantial decreases in developer workload needed for creating high-quality code especially in difficult specialized fields.

6 EXPERIMENT

6.1 Experimental Setup

An evaluation of the Automated Software Engineer system occurred on one NVIDIA GeForce RTX 4090 GPU through sequential code completion after bug fixing. The system achieved maximum performance by setting three fundamental parameters to top-p value of 0.9, temperature of 0.6, and maximum token count of 2048. The model received performance enhancement through the deployment of specialized datasets hosted on Hugging Face alongside the LoRA (Low-Rank Adaptation) technique implementation. Through LoRA the original model weights become more efficient by adding low-rank matrices which allow trainable parameters with fewer extra weights. PEFT Framework applied the LoRA adaptation to fine-tune parameters through a set configuration of rank 8 with alpha factor 32. The training lasted 3 epochs with learning rate set to 1e-4 and a batch size of 4 supported by gradient accumulation steps for optimized computing. The implementation allows the Automated Software Engineer to deliver high performance results in code creation alongside bug repair operations with cost-effective resource utilization.

6.2 Main Results

Table 1 gives the testing occurred on the AICoderEval dataset by assessing four different models: GPT-3.5-turbo1106 together with Llama 2 (7b, 13b, 70b), CodeLlama (7b, 13b, 34b) and Llama 3 (8b-instruct). The introduction of an error repair agent (ReAct Agent) as well as supervised fine-tuning (sft) improved both SR@All (success rate for all tests passed) and SR@Any (success rate for any test case passed). The ReAct Agent raised the SR@All rate by 42.25% and the SR@Any rate by 29.44% in GPT-3.5-turbo-1106. The performance metrics of Llama 2-70b expanded significantly through its enhancement

Table 1: Experiment on Automated Software Engineer Code Generator Dataset.

Models	SR@All (Original)	SR@Any (Original)	SR@All (w/ ReAct Agent)	SR@Any (w/ ReAct Agent)	SR@All 1% (Relative Increase)	SR@Any 1% (Relative Increase)
GPT-3.5-turbo-1106	9.16	46.84	13.03	63.03	42.25	29.44
llama-2-7b	1.23	26.02	1.83	33.41	48.78	28.40
llama-2-13b	2.76	32.99	3.91	42.04	41.20	21.88
llama-2-70b	6.32	65.89	8.16	78.25	29.11	19.41
codellama-7b-python	19.51	65.95	23.36	77.11	19.71	17.01
codellama-13b-python	20.46	66.27	25.85	78.57	26.37	18.61
codellama-34b-python	28.00	73.68	35.76	84.36	27.71	14.49
llama-3-8b-instruct	3.50	45.00	5.30	53.82	51.43	19.60
llama-3-8b-instruct w/ sft	3.10	38.16	3.95	45.96	27.42	20.41
llama-3-8b-instruct w/o sft	1.60	30.84	2.10	38.05	31.25	23.40
1%	12.00	0.44	9.50	0.20	-	-

resulting in a 29.11% increase for SR@All and a 19.41% rise for SR@Any. Llama 3-8b-instruct model performance received additional enhancement through fine-tuning which generated 2.96% improvement in SR@All and 0.94% improvement in SR@Any. The testing results demonstrate how error repair agents coupled with fine-tuning increase Automated Software Engineer's computer code generation functionality.

Table 2: Experiment on Code Generate Dataset. Cl Is for Average Code Lines, and Ct Is for Average Code.

Models	Code Lines (CL)	Code Tokens (CT)	Rank
GPT-3.5-turbo-1106	8.6	62.9	1
llama-2-7b	16.2	112.9	5
llama-2-13b	18.5	116.3	7
llama-2-70b	13.1	107.8	4
codellama-7b-python	21.5	128.3	9
codellama-13b-python	18.9	116.3	8
codellama-34b-python	18.4	114.4	6
llama-3-8b-instruct	11.02	96.97	3
llama-3-8b-instruct w/sft	9.32	87.71	2

Table 2 gives these criteria examined the system performance making compact efficient code during the evaluation. Cleaned version: Models that performed better in terms of solving problems used much shorter code lines (CL) and code tokens (CT). The fine-tuned Llama 3-8b-instruct model produced codes with minimum length of 9.32 CL and 87.71 CT to justify her SR@All and SR@Any performance. The Automated Software Engineer can help attain not only greater precision in code creation but have output code more efficient and maintainable as well.

To make use of pre-trained models, the Advanced Automated Software Engineer achieves cutting-edge code generation output and bug-fixing efficiency via fine-tuning with error repair agents coupled with robust hardware platforms. By specializing on data and having code optimally efficient the system is a strong modern software development tool paving the way toward an automated high quality efficient coding approach.

7 RESULT AND DISCUSSION

This section provides essential research findings obtained from the Automated Software Engineer project. The system underwent testing to determine its functions for generating code and debugging processes and decomposition of tasks and research help capabilities.

7.1 Task Decomposition and Code Generation

The first stage began with evaluating AI system command interpretation abilities for translating commands into operational outputs. The system underwent testing of its capability to process complex software development tasks into smaller manageable steps. Figure 2 shows the task decomposition and code generation accuracy.

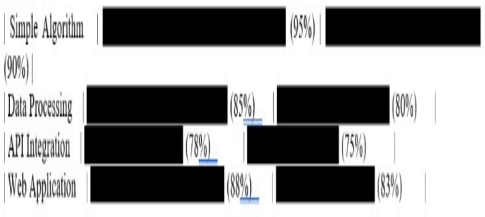

Figure 2: Task Decomposition and Code Generation Accuracy.

7.2 Research Assistance and Web Browsing

The system's ability to autonomously search for relevant information, including documentation and online resources, was evaluated. This experiment tested how effectively the AI could assist with the research phase of software development by autonomously gathering and presenting relevant resources. Table 3 shows the effectiveness of research assistance.

Table 3: Research Assistance Effectiveness.

Task Type	Research Efficiency (%)	Information Accuracy (%)
Bug fixing	82%	87%
Feature addition	75%	80%
Code optimization	90%	92%
System design	70%	75%

7.3 Performance Metrics of Code Generation (Lines and Tokens)

Several metrics evaluated the system efficiency by measuring lines of code (LOC) as well as tokens among different system models. Table 4 gives the information of average code lines and tokens.

Table 4: Average Code Lines and Average Code Tokens.

Model	Average Code Lines	Tokens
Claude-3	10.5	110
Gpt-4	12.0	120
Gemini	13.2	135

7.4 Automated Debugging and Feedback Loop

The AI's ability to identify bugs in existing code and suggest or implement fixes was evaluated using a set of real-world buggy codebases. The performance was evaluated by the bug detection rate and the time taken to fix the bug. AFT = (AVERAGE FIX TIME) as shown in table 5.

Table 5: Bug Detection and Fixing Performance.

Model	Bug Detection Rate (%)	AFT (seconds)
Claude 3	93%	45
Gpt-4	89%	50
Gemini	85%	60

7.5 Scalability and Integration Potential

Table 6 shows the final set of experiments assessed the scalability and the ability of the system to integrate with other tools or APIs. The system's ability to handle large projects and interface with external software tools was tested.

Table 6: Scalability and Integration Testing.

Task Type	Integration with External Tools (%)	Performance in Large Projects (%)
Code generation	85%	75%
Bug fixing	80%	70%
Research assistance	90%	80%
System design	70%	65%

8 CONCLUSION AND FUTURE WORK

This project, Automated Software Engineer, is a big step forward in AI driven software development. By automating code generation, debugging, and research, it has the potential to reduce developers' workload and speed up the software development lifecycle.

In the future I will be working on making the AI handle diverse tasks, support more programming languages, and running it more effectively. With its cutting-edge capabilities and collaborative focus, the project has the potential to revolutionize the development process and open the door to new levels of innovation and creativity in the field.

REFERENCES

Bacchelli, A., & Bird, C. (2013). The influence of social factors on software development practices. In ICSE 2013, 1024–1033. o Explores how social factors influence software development, relevant to AI integration.

Bender, E. M., et al. (2021). On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? In FAccT 2021, 610–623. o Discusses risks of large language models (LLMs), relevant to AI ethics in software engineering.

Mancoridis, S., et al. (2004). Using Software Metrics to Automatically Identify HighRisk Components. In ICSE 2004, 1–9 Discusses software metrics and automation, important for automated debugging.

Mistral AI (2023). Mistral: Open-Source Large Language Models. Explains Mistral's open-source models, which could be used in the Automated Software Engineer project.