**AI-Driven Bug Detection and Resolution in Code Using Large Language Models**

## 

**Table Of Content**

[**1. Introduction 3**](#_tzb0sc4ia43z)

[1.1 Aim and Objectives 3](#_hnlgozn7l30x)

[1.2 Research Question 3](#_jg756g203ubq)

[**2. Background and Justification 3**](#_odrhf4sb8q52)

[**3. Proposed Artifact and Societal Impact 5**](#_d4t1mhdqyt7d)

[**4. Resources and Project Implementation 8**](#_ases2kns0lpo)

[4.1 Resources 8](#_ij750n7wsurh)

[4.2 Project Implementation 9](#_hbqf7z1ji5cq)

[**5. Conclusion 11**](#_yazh42rao73u)

[**References 12**](#_q7lvl08az1jh)

## **1. Introduction**

But, as these systems get larger, they have some those things which make it annoying to find out the defects and remove them, even exhausting to the developers. Contrary to their expectations, using manual code review or utilizing bin/discrete Static Code Analysis techniques as an approach is only slightly effective in detecting a plethora of subtle faults and can at times be dangerous; however they waste development cycles. Recently, there has been the emergence of deep learning models like LLMs and they primarily aid programmers by assisting in the development process, code creation, or even bug detection. This scholarly work deals with a development of an AI-based system that can incorporate the usage of LLMs in searching and correcting bugs in software code on an over-arching quest of improving code quality, and by extension, productivity of developers.

### **1.1 Aim and Objectives**

The primary aim of this research project is to develop an AI-driven bug detection and resolution system using LLMs to assist developers in identifying and fixing bugs in software code. The specific objectives are as follows:

1. Search for a set of code samples covering a range of bug types is found in repositories that are freely available to the public.
2. Train the obtained LLM greater focus on the bug-related data collected from system logs to make it more suitable for bug identification and rectification purposes.
3. Create a paring model that will take code snippets as input, identify possible bugs from the input, and return corrected code.
4. Enhance the bug detection and resolution system to embed into a development environment which will be useful to a developer as a plugin.
5. Assess the capability and efficiency of the system by defining valid standards and employing practical examples to test its feasibility.

### **1.2 Research Question**

The primary research question guiding this project is: Can an AI-based system utilizing Large Language Models effectively detect and resolve bugs in software code, thereby improving code quality and developer productivity?

## **2. Background and Justification**

Due to an exponential expansion in the size and structure of software systems, the process of detecting and rectifying mistakes in a system has become cumbersome and tedious for application specialists. Incorporating a testing method designed to prey upon bugs is more optimal than typical bug detection methods, which include manual code inspections, static analysis, and penetration testing, which often yield suboptimal results by either missing or taking a long time to identify some of the most basic and subtle bugs.

The development of intelligent methods in the recent past has helped in incorporating innovative solutions to assist developers in the overall field of software engineering in general. One such advancement is the concept of Large Language Models or LLMs, which show a great deal of potential in the generation and comprehending of natural text that is as good as human authored text, including computer code. A large range of LLMs including mistral 7b, GPT-4, Codex etc have now demonstrated capabilities in terms of code completion, code summarization, and even bugs detection and their fixes.

In the domain of software engineering, some precedented researches have been dedicated to examine the possibility of LLMs. Sarkar et al. (2022) provided a detailed analysis and synthesis of the application of AI for programming based on their study on the perception of using AI in programming including both the opportunities and shortcomings encountered in implementing AI for programming. Based on their research, they recommend for the use of Artificial Intelligence programming aid to developers with an aim of increasing their productivity of work and the quality of code developed while at the same time, they point towards the correct approach to the development of these systems as being important towards the aimed effectiveness of such a non-trivial undertaking.

In their study, carried out in 2024, Taghavi and Feyzi prioritized and solely focused on the use of LLMs in identifying and managing software flaws and cyber security risks. LLMs can have practical application as it can show potential hazards that may exist in code and recommend how they should be mitigated to improve the overall security of the software. This shows that the benefits of LLMs are not restricted to the developers themselves, how they work, and their efficiency, but can also help to enhance the overall quality and security of the software applications generated.

In an empirical study, Yetiştiren et al. (2023) assess the code quality of AI-enhanced code automation tools as offered by GitHub Copilot AI Pair Gitpod ChatGPT and Amazon CodeWhisperer. It has been established by researchers that such tools as the four discussed in this paper may develop functionally correct code, there is still an issue of readability, maintainability and efficiency in the developed code. This illustrates the need to not only develop code but also to develop code that is as optimal as possible and in accordance with the appropriate standards.

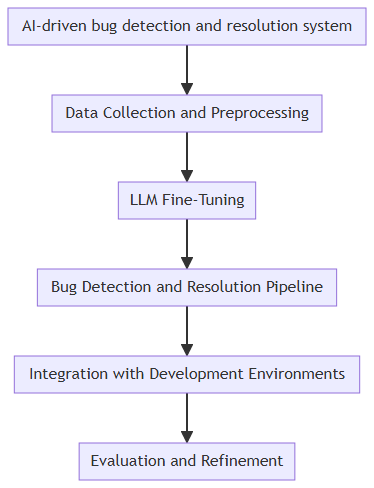
In this particular case, it was found that though there is a developing literature about AI supported software engineering, there is still a gap in having a general system which can utilize LLMs in enhancing the detection and identification of bugs. Previous work has mainly worked towards assessing the effectiveness of LLMs on different levels or in the context of certain tasks, while the work that has been done to build an end-to-end system utilizing LLMs to aid the search process of developers in locating and correcting program bugs has not been conducted extensively.

To this end, this work proposes a novel approach of an AI-enabled bug detection and resolution feeding off LLMs for detection of bugs in code as well as offering corrected code. By utilizing LLMs in the SDLC, this system will have measurable benefits of enhancing the quality of the code produced and providing quicker detection and remediation of bugs, as well as increase developer efficiency and efficacy.

For this, the proposed system will use the logical and mathematical capabilities of state-of-art LLMs like Mistral 7b that are pretrained on abundant code and natural language data. These models will be further trained on a customized set of code snippets with different kinds of bugs and the correct versions of such code. Such fine-tuning is going to help LM generate specialized autodidacts that can focus on bug detection and resolution only which remarkably helps in identifying even the small bugs and correcting them and providing a proper code suggestion.

## **3. Proposed Artifact and Societal Impact**

The primary artifact resulting from this research project will be an AI-driven bug detection and resolution system that seamlessly integrates with popular development environments as a plugin. This system will leverage state-of-the-art LLMs like Mistral 7b to analyze code snippets provided by developers, identify potential bugs, and suggest corrected code as output.



The proposed system will consist of several key components:

1. Data Collection and Preprocessing: The body of code samples will be diverse and collected from various programming languages and programming domains, originating from resources such as GitHub. Examples of these codes will be well documented providing detailed information about the existing bugs and the corrected versions. The data collected will then in turn need to go through preprocessing steps in order to clean it up and prepare it to be used to later train the LLMs.
2. LLM Fine-Tuning: The datasets to fine-tune the pre-trained LLMs will be a carefully selection of buggy and corrected code samples. At some point during these rounds, there will be fine-tuning, which will make the LLMs more specialized in the game of bug identification and fixing since they will be able to identify even the most complicated bugs as well as offer code suggestions. Therefore, fine tuning, which is the process of changing the parameters of the models and training the LLMs for the particular task using techniques such as transfer learning and domain adaptation, will be required.
3. Bug Detection and Resolution Pipeline: Cambridge is sure to build a strong pipeline to handle the received code sub-strings, analyze possible bugs with the help of fine-tuned LLMs, and generate fixed code options. This pipeline will cater for the flow of data beginning from the stage where the input code is received to the ultimate stage of coming up with the corrected code. That will involve code parsing, applying abstract syntax tree to do the analysis, and natural language processing to get the right understanding or manipulate the code.
4. Integration with Development Environments: Future bug detection and resolution will be developed as a premium plugin that smoothly fits major environments like Visual Studio Code. These two components will enable the developers to get access to the functionalities of the system without the need for shifting to other application screens or using other applications and tools by integrating into the developers’ integrated developmental environment. The idea of the plugin is to have an input box where the developer enters his/her code and gets the results of bug detection; he/she can also see the corrected code here.
5. Evaluation and Refinement: There will be a validation and optimization of the AI-based bug detection and its fixing performance by analyzing certain parameters like accuracy, precision, recall, and F1-score. To verify the effectiveness of the system at recognizing bug patterns and finding and fixing bugs, then the system will be run on a held-out dataset of code samples. In fact, the next step that has to be performed is the ongoing evaluation and adjustment of the system based on what has been learned throughout the process.

The adopted artifact for the proposed project will seek to appreciate the use of AI and LLMs in software engineering where the use of AI in coding will be promoted. This system would go a long way in assisting a developer in identifying the bugs that exist in the system and help come up with solutions to fix them saving a lot of time since fixing the bugs would be automated.

It could thus be seen that this research is relevant in several ways and is fit to impact society in various ways. In the first place, the enhancement of the quality of code and, consequently, the time and efforts necessary to detect bugs and eliminate them, which the developed system suggests, may contribute to the creation of more reliable and secure software applications. In contemporary society,Computerization of various goods and services and the development of software systems for critical industries has become a notable theme, thus; the need t enhance secuirty of the respective systems is of considerable importance. This system, which is effective in early identification and solving of bugs, offers an efficient means to contain the dangerous outcomes dependent on software vulnerabilities which include data leaks, system crashes, and cyber-attacks.

Besides the impact on the category of people that constitutes software developers and software organizations, the proposed system can imply more extensive societal impact. By contributing to the development of more secured and effectual software applications, this system will serve towards developing confidence and faith in new technology. Therefore, trustworthiness and reliability of the systems that are being developed and are being used become a critical component as the society reliefs on technology for basic needs such as healthcare, finance, and transportation. The successful implementation of the AI-based bug detection and automatic rectification system can go a long way helping in improving the sustainability, security, and dependability of digital systems; which in turn helps in making people more confident about using technology.

In addition, the presented R&D effort in the development of this AI-based system can also result in further progress in the area of AI-enhanced software engineering. These insights derived from this work may help create an avenue for future research and development of AI and LLMs in the field of software development. Of course, many tips can be given to other designers and developers of such systems, as well as the assessment results can be helpful in the future development of more effective AI-assisted coding tools.

The societal impact of this research project extends beyond the software development industry, as it can contribute to creating more secure, reliable, and inclusive digital solutions. The democratization of coding assistance, the economic benefits of reduced bug fixing costs, and the broader implications for building trust in digital technologies underscore the far-reaching impact of this research project. As such, this project represents a significant step forward in the field of AI-assisted software engineering, paving the way for more intelligent and efficient tools to support developers in their daily work and ultimately benefiting society as a whole.

## **4. Resources and Project Implementation**

To successfully implement the AI-driven bug detection and resolution system, a range of resources and a well-defined project plan are essential. This section outlines the necessary resources, including hardware, software, and datasets, as well as the key steps involved in the project implementation.

### **4.1 Resources**

1. Hardware:
   * High-performance computing systems: As to acquire and optimize the LLMs and to carry out the model inference, a high-end computing resource is necessary. This may include GPU clusters or cloud computing platforms that would be used in order to solve computational needs that arise whenever working with large - scale language models.
   * Storage systems: Sufficient memory space should be provided for storing source code datasets, initial data and models received after that, as well as for results of each stage of the work during the overall process of the project.
2. Software:
   * Programming languages: The project will mainly focus on developing algorithms and models in the field of AI and NLP, and Python is a language commonly used in this industry. Python has a vast array of libraries and frameworks that facilitate the creation of automata and intelligence systems.
   * AI frameworks and libraries: To engage with LLMs and put into practice the bug detection and resolution chain, famous AI structures and libraries including Tensor Flow, Py Torch, and Mmistral 7b shall be used. Such tools offer all the essentials required for the training, fine-tuning, and deployment of the LLMs.
   * Development environments: IDE like: Visual Studio Code will be utilized for writing the code, debugging and test the code as well. Such IDEs are useful in designing a fertile and efficient zone for putting into practice the multifaceted parts of the system.
   * Version control systems: Git will be used as a tool in order to manage different versions of the project and History & Timeline will be used as a tool for organizing the work of developers and co-ordinators. It will assist in observing the variations, administering and enhancing branches and also help coordinate work in teams all through the project.
3. Datasets:
   * Open-source code repositories: As a result of the fact that it is easier to obtain more data from various sources where developers share their code, open-source repositories that include GitHub will be used. These repositories actually store a HOUGE number of samples of actual code written from different programming languages and domains.
   * Bug datasets: Additional data that is already available and can be helpful includes existing bug datasets comprising of code examples, for instance, BugSwarm dataset and Defects4J dataset. These datasets correspond to a collection of buggy code and pairs of buggy and fixed code containing annotations that can be used to evaluate the bug detection and resolution system.

### **4.2 Project Implementation**

The project implementation will follow a structured approach, consisting of several key steps:

1. Data Collection and Preparation:
   * Code sample collection: It will be initially collecting a large-scale dataset of code samples which is available in the open-source repository. This will entail searching through these repositories to obtain code snippets, and further working on the above-mentioned criteria: languages used in the snippets, domains of the applications the snippets belong to, and the bugs targeted by the snippets.
   * Data annotation: These collected code samples will be manually analyzed and some of the bugs present in the code as well as the corrected version will also be noted. Regarding the process of annotation we anticipate that this will be done in conjunction with the developers who are experienced and have some software engineering background so that to make sure that the annotations are correct and accurate.
   * Data preprocessing: These will be pre-processed to drop, convert and rearrange them into proper format for analysis and modeling. Some such pre-processor actions may be to clear out non-essential commentaries in the code structure, format the code to provide a semblance of order in coding arrangement, and finally the division of data into the training set, the validation set and the test set.
2. Model Development and Training:
   * LLM selection and fine-tuning: A pre-trained LLM like Mistral 7b will be used in this work to form the basis of the bug detection and resolution system. After collecting and annotating proper data, the LLM will be trained on the collected and annotated dataset using transfer learning principles. This fine-tuning process will enable the model to become adapted for the specific task of bug detection and demise.
   * Hyperparameter tuning: Specific parameters of the LLM, to name learning rate, the size of the batches, as well as the number of the epoch of training will be fine-tuned. This shall involve running tools and testing of the model on a validation set with a view of determining the right hyperparameters to use.
   * Model training and validation: The fine-tuned LLM will thereafter be trained on the prepared data set using correct training techniques like mini-batch gradient descent. The autoencoding model will be regularly evaluated with respect to the validation set to observe the learning condition and consider potential overfitting/underfitting problems.
3. Bug Detection and Resolution Pipeline:
   * Pipeline design: An intricate pipeline will be laid down which will entail receiving input code snippets, passing them through the training of the LLM to detect bugs, and then give corrected code recommendations. This pipeline will involve numerous stages and sub-stages including the code parsing stage, abstract Syntax tree analysis stage and also natural language processing techniques stage.
   * Code parsing and representation: The input code snippets will be pre-processed to convert it into a convenient form such as Abstract Syntax Tree or list of tokens which can be easily feasible for the LLM. This representation will cover all the features of the code including their structural and semantic aspects.
   * Bug detection: The trained LLM will then scan through the translated code representation for errors as hinted by the Parse Tree. The model will then apply what it has learnt by identifying similarities and differences that may point at bugs such as syntax errors, contradicting logical sequences, or programmed securities threats.
   * Code correction: Moreover, when the bugs are found, the LLM will be able to produce the code corrections as the examples. This will entail feeding to the model the buggy code, and then using the generated language to suggest fixes or changes to it. The generated suggestions will target at addressing the bugs found as well ensuring that the functionality and style of code is not affected.

## **5. Conclusion**

This Pilot Study Report describes an AI based bug detection and resolution system which uses LLM technology to aid software developers Identify and fix bugs. In particular, the proposed system will eliminate the need for external tools and allow for the immediate detection and fixing of bugs, which will increase the overall quality of the code and reduce the time developers spend on this task. It will add to the existing literature about integrating AI in software development and show that LLMs can prove useful when it comes to solving diverse coding issues. As such, the outcome and the positive effect that it will inevitably have on society will be capable of significantly reshaping the overall software creation process and ultimately improving the existing environment.

## **References**

Baumgartner, N., Padma Iyenghar, Padma Iyenghar, Schoemaker, T. and Pulvermüller, E. (2024). AI-Driven Refactoring: A Pipeline for Identifying and Correcting Data Clumps in Git Repositories. Electronics, 13(9), pp.1644–1644. doi:https://doi.org/10.3390/electronics13091644.

Beheshti, A. (2024). Natural Language-Oriented Programming (NLOP): Towards Democratizing Software Creation. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2406.05409.

Ciniselli, M., Puccinelli, N., Qiu, K. and Di Grazia, L. (2024). From Today’s Code to Tomorrow’s Symphony: The AI Transformation of Developer’s Routine by 2030. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2405.12731.

Fan, A., Beliz Gokkaya, Harman, M., Mitya Lyubarskiy, Sengupta, S., Yoo, S. and Zhang, J.M. (2023). Large Language Models for Software Engineering: Survey and Open Problems. arXiv (Cornell University). doi:https://doi.org/10.48550/arxiv.2310.03533.

Fu, M., Pasuksmit, J. and Tantithamthavorn, C. (2024). AI for DevSecOps: A Landscape and Future Opportunities. arXiv (Cornell University). doi:https://doi.org/10.48550/arxiv.2404.04839.

Hao, S., Shi, X. and Liu, H. (2024). Exploring the Potential of Pre-Trained Language Models of Code for Automated Program Repair. Electronics, 13(7), pp.1200–1200. doi:https://doi.org/10.3390/electronics13071200.

Ilame, N. (2024). Machine Learning-Powered Programming: Exploring the Fusion of AI and Coding. Innovative Computer Sciences Journal, [online] 10(1), pp.1−11–1−11. Available at: https://innovatesci-publishers.com/index.php/ICSJ/article/view/16 [Accessed 16 Jun. 2024].

Jiang, J., Wang, F., Shen, J., Kim, S. and Kim, S. (2024). A Survey on Large Language Models for Code Generation. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2406.00515.

Johnsson, N. (2024). An In-Depth study on the Utilization of Large Language Models for Test Case Generation. [online] www.diva-portal.org. Available at: https://www.diva-portal.org/smash/record.jsf?pid=diva2:1835852 [Accessed 16 Jun. 2024].

Klemmer, J.H., Horstmann, S.A., Patnaik, N., Ludden, C., Burton Jr, C., Powers, C., Massacci, F., Rahman, A., Votipka, D., Lipford, H.R., Rashid, A., Naiakshina, A. and Fahl, S. (2024). Using AI Assistants in Software Development: A Qualitative Study on Security Practices and Concerns. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2405.06371.

Noever, D. (2023). Can Large Language Models Find And Fix Vulnerable Software? [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2308.10345.

Nygård, J. (2024). AI-assisted code generation tools. [online] laturi.oulu.fi. Available at: https://oulurepo.oulu.fi/handle/10024/50546 [Accessed 16 Jun. 2024].

Omari, S., Kshitiz Basnet and Wardat, M. (2024). Investigating large language models capabilities for automatic code repair in Python. Cluster computing. doi:https://doi.org/10.1007/s10586-024-04490-8.

Rasheed, Z., Sami, M.A., Waseem, M., Kemell, K.-K., Wang, X., Nguyen, A., Systä, K. and Abrahamsson, P. (2024). AI-powered Code Review with LLMs: Early Results. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2404.18496.

Sandoval, G., Pearce, H., Nys, T., Karri, R., Garg, S. and Dolan-Gavitt, B. (2023). Lost at C: A User Study on the Security Implications of Large Language Model Code Assistants. [online] www.usenix.org. Available at: https://www.usenix.org/conference/usenixsecurity23/presentation/sandoval [Accessed 16 Jun. 2024].

Sarkar, A., Gordon, A.D., Negreanu, C., Poelitz, C., Ragavan, S.S. and Zorn, B. (2022). What is it like to program with artificial intelligence? arXiv:2208.06213 [cs]. [online] Available at: https://arxiv.org/abs/2208.06213.

Sri, H. and Ambati (2023). Security and Authenticity of AI-generated code. [online] Available at: https://harvest.usask.ca/bitstream/handle/10388/15154/AMBATI-THESIS-2023.pdf?sequence=1 [Accessed 16 Jun. 2024].

Taesiri, M.R., Macklon, F., Wang, Y., Shen, H. and Bezemer, C.-P. (2022). Large Language Models are Pretty Good Zero-Shot Video Game Bug Detectors. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2210.02506.

Taghavi, S.M. and Feyzi, F. (2024). Using Large Language Models to Better Detect and Handle Software Vulnerabilities and Cyber Security Threats. [online] Research Square. doi:https://doi.org/10.21203/rs.3.rs-4387414/v1.

Tang, D., Chen, Z., Kim, K., Song, Y., Tian, H., Ezzini, S., Huang, Y., Klein, J. and Bissyande, T.F. (2024). CodeAgent: Collaborative Agents for Software Engineering. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2402.02172.

Widjojo, P. and Treude, C. (2023). Addressing Compiler Errors: Stack Overflow or Large Language Models? [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2307.10793.

Wu, Y., Jiang, N., Pham, H.V., Lutellier, T., Davis, J., Tan, L., Babkin, P. and Shah, S. (2023). How Effective Are Neural Networks for Fixing Security Vulnerabilities. arXiv (Cornell University). doi:https://doi.org/10.1145/3597926.3598135.

Yang, Z., Sun, Z., Yue, T.Z., Devanbu, P. and Lo, D. (2024). Robustness, Security, Privacy, Explainability, Efficiency, and Usability of Large Language Models for Code. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2403.07506.

Yao, Y., Wang, J., Hu, Y., Wang, L., Zhou, Y., Chen, J., Gai, X., Wang, Z. and Liu, W. (2024). BugBlitz-AI: An Intelligent QA Assistant. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2406.04356.

Yetiştiren, B., Özsoy, I., Ayerdem, M. and Tüzün, E. (2023). Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT. arXiv:2304.10778 [cs]. [online] Available at: https://arxiv.org/abs/2304.10778.