**BUG DETECTION AND FIX GENERATION**

**ABSTRACT**

This project proposes an AI-powered system for automated bug detection and fix generation using the CodeLlama model. Leveraging multi-task classification, the system predicts the **bug type**, **severity**, and **error line number** from source code written in **Java, Python, C, and C++**. A secondary generative model then provides context-aware fixes for the detected issues. We employ **4-bit quantization** for efficiency and integrate a **Gradio-based UI** to allow users to interact with the system seamlessly. The proposed solution aims to reduce debugging time, increase code reliability, and bring AI-driven development tools closer to production environments.

**Keywords:** bug detection, bug fix generation, multitask classification, large language models (LLMs), GenAI, transformer-based architecture, software debugging

**INTRODUCTION**

Software development is a complex process, and one of its most time-consuming and error-prone phases is debugging. Traditional debugging methods often require manual inspection, which can lead to delays, increased costs, and potential oversights in large-scale codebases. To address these challenges, we propose an AI-powered system that automates **bug detection** and **bug fix generation**, utilizing state-of-the-art transformer-based architectures.

Our system is built upon the **CodeLlama** model, fine-tuned to perform **multi-task classification** for identifying:

* The presence of a bug (binary classification),
* The **type of bug** (e.g., syntax, logical, runtime),
* Its **severity level** (e.g., critical, major, minor),
* The **error line number** in the source code.

We support code written in popular programming languages including **Java**, **Python**, **C**, and **C++**, ensuring broad applicability across software projects. To complement the classification model, we incorporate a **generative model** that provides **context-aware code fixes** for the identified bugs. This two-stage approach enables not only the detection of errors but also the automated suggestion of appropriate corrections, enhancing developer productivity and code quality.

To ensure efficiency, especially when working with resource-constrained environments, we employ **4-bit quantization** techniques. Additionally, the system features an intuitive **Gradio-based user interface**, enabling users to interact with the model seamlessly by inputting code snippets and receiving real-time bug analysis and suggested fixes.

Our solution aims to:

* **Reduce debugging time** significantly,
* **Improve code reliability**

**PROBLEM STATEMENT**

Debugging is a critical yet tedious part of the software development lifecycle. Developers often spend a significant portion of their time identifying and resolving bugs, which can slow down the development process and lead to increased costs. Traditional debugging tools rely heavily on static analysis and manual inspection, offering limited support for understanding the context of the bug or providing actionable fixes. These limitations become more evident in large, complex codebases and across diverse programming languages.

With the rise of intelligent code understanding through large language models (LLMs), there is a growing opportunity to automate the debugging process. However, existing AI-based solutions either focus solely on bug detection or lack multi-language support and fine-grained predictions such as bug severity, type, and location. This creates a clear need for a comprehensive, AI-driven system that can accurately detect bugs, classify their severity and type, localize the error line, and suggest meaningful fixes — all within a unified and efficient framework.

**OBJECTIVES**

* Develop an AI-driven bug detection model using CodeLlama.
* Implement multi-task classification to predict bug type, severity, and error location
* Optimize the model with 4-bit quantization for memory efficiency.
* Design a bug-fix generation pipeline to provide corrected code suggestions.
* Create an interactive Gradio UI for users to submit and analyze their code.

**LITERATURE SURVEY**

Traditional debugging methods, including manual inspection and static or dynamic analysis tools, have long served as the backbone of software maintenance. Manual debugging is labor-intensive and susceptible to oversight, while static analysis tools like SonarQube and Pylint focus primarily on syntax and style errors without understanding the deeper context of code logic. Dynamic debugging tools such as GDB and Valgrind require code execution and are often inefficient when dealing with large-scale or early-stage codebases, limiting their practicality in real-time development environments.

Recent advances in AI and large language models have introduced new possibilities for automated code understanding and bug detection. Models like CodeBERT and GraphCodeBERT leverage transformer architectures to perform code classification and summarization tasks effectively, though they fall short in generating actionable fixes. Similarly, tools like DeepCode and Infer apply AI-enhanced static analysis but do not provide real-time code corrections. To overcome these gaps, CodeLlama, a code-specialized LLM offers promising capabilities for both understanding and generating code. When combined with 4-bit quantization, it enables efficient deployment without compromising performance.

**MODEL ARCHITECTURE**

**UI**

**Gradio Interface**

**Bug Detection**

**CodeLlama model**

**model**

**4-Bit quantization to reduce**

**Memory usage**

**Multi task**

**classification**

**Bug Report**

* **Bug type**
* **Bug severity**
* **Error line number**

**Bug Fix**

**On Java, Python, C**

**& C++**

**METHODOLOGY**

**Data Collection & Preprocessing**

* Created a hybrid dataset by combining:
  + Real-world code snippets collected from GitHub, Stack Overflow, and open-source repositories.
  + Synthetic buggy code samples generated using **prompt engineering** with large language models.

**Model Training & Quantization**

* Fine-tuned the **CodeLlama** model for **multi-task classification** on the labeled dataset.
* Used bitsandbytes library to apply **4-bit quantization**, reducing the model size and memory footprint for efficient deployment.

**Bug Detection Pipeline**

* **Input**: Raw code snippet (Java, Python, C, C++)
* The fine-tuned model predicts:
  + Bug Type (Syntax, Runtime, Logical)
  + Bug Severity (Low, Medium, High, Critical)
  + Error Line Number

**Bug Fix Generation**

* **Input**: Generated bug report from the detection model
* Utilizes CodeLlama’s code generation capabilities to produce:
  + **Context-aware fixes** tailored to the detected bug type and location

**UI Implementation**

* Developed an **interactive Gradio-based UI**
* Ensures real-time usability for developers and seamless interaction with the AI system

**REFERENCES**

* Meta AI. (2023). **Code Llama: Open Foundation Models for Code**. [Online]. Available: <https://ai.meta.com/blog/code-llama-large-language-models-for-code>
* Dettmers, T., Zettlemoyer, L., Lewis, M., & Yih, W. (2022). LLM.int8(): **8-bit Matrix Multiplication** for Transformers at Scale. *arXiv preprint arXiv:2208.07339*.
* Allamanis, M., Barr, E. T., Bird, C., & Sutton, C. (2018). **A Survey of Machine Learning for Big Code and Naturalness**. *ACM Computing Surveys (CSUR)*, 51(4), 1-37.
* Pradel, M., & Sen, K. (2018). **DeepBugs:** A Learning Approach to Name-Based Bug Detection. *Proceedings of the ACM on Programming Languages*, 2(OOPSLA), 1-25.
* Raychev, V., Vechev, M., & Krause, A. (2015). **Predicting Program Properties** from "Big Code". *ACM SIGPLAN Notices*, 50(1), 111-124.
* Bader, J., & Lopes, C. V. (2019). When Deep Learning Met Static Analysis: Towards a Fully **Automated Method for Detecting Software Bugs**. *IEEE Transactions on Software Engineering*.
* Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., ... & Zaremba, W. (2021). **Evaluating Large Language Models** Trained on Code. arXiv preprint arXiv:2107.03374.
* Jain, A., Aggarwal, G., & Sankaranarayanan, K. (2023). BugLab: Learning **to Find Bugs from Human-Written Programs** with Weak Supervision. arXiv preprint arXiv:2306.14830.
* Karampatsis, R. M., & Sutton, C. (2020). How Often Do Single-Statement Bugs Occur? The **ManySStuBs4J Dataset**. Empirical Software Engineering, 25, 2798–2838.
* Tufano, M., Watson, C., Bavota, G., White, M., Poshyvanyk, D., & Lucia, A. D. (2019). An Empirical **Study on Learning Bug-Fixing Patches** in the Wild via Neural Machine Translation. ACM Transactions on Software Engineering and Methodology (TOSEM), 28(4), 1–29.
* Puri, R., Patil, H., Sreedhar, V., & Suh, G. E. (2021). **LLM4Code**: Large Language Models for **Code Understanding and Generation**. arXiv preprint arXiv:2110.12841.