Bug Detection and Fixing System

Developed By: Shruti Gorgile, Shubhangi Takbide, Shruti Kubade

Guided By: Ms. Nitu. L. Pariyal

Project Title:

Build a machine learning model that can automatically identify bugs (or potential errors) in a given piece of code suggest fixes. This project requires designing, training and evaluating an ML model that can parse source code. classify bug types and generate fix.

Project Overview

Objective:

To develop a Machine Learning-based system that can automatically identify bugs in source code, classify them into specific categories, and suggest potential fixes, enhancing the efficiency and accuracy of the debugging process.

Step 1: Define Project Objectives and Requirements

Objectives:

1. Automated Bug Detection – Identify errors in source code.
2. Bug Classification – Categorize detected bugs.
3. Fix Generation – Suggest corrections for identified bugs.
4. Efficient Debugging – Reduce manual debugging efforts.
5. Scalability – Support multiple programming languages.

Requirements:

- Data: Buggy code snippets and fixed versions.
- Tools: Python, PyTorch, Streamlit, Fast API, Transformers.
- Infrastructure: GPU for training, IDE (PyCharm).

Step 2: Setup Development Environment

1. Install Required Tools & Libraries

- Python (3.x) – Core programming language

- Libraries:
- pip install streamlit, torch, transformers, fast API, request, accelerate.
- IDE: PyCharm
- Version Control: GitHub

2. Configure Environment

- Set up a virtual environment:
- python -m venv bug-fix-env
- source bug-fix-env
- bug-fix-env # (Windows)
- Enable GPU acceleration

3. Dataset Preparation

- Collect buggy and fixed code datasets from open-source repositories (e.g., GitHub, Bugzilla).
- Preprocess data using tokenization and Abstract Syntax Trees (AST).

Step 3: Using a Pretrained Model for Bug Detection and Fixing

1. Model Overview: CodeLlama is an advanced large language model optimized for understanding and generating code.
2. Installation: Requires libraries like transformers, torch, and accelerate.
3. Loading the Model: CodeLlama can be easily loaded using Hugging Face's AutoModelForCausalLM and AutoTokenizer.
4. Bug Detection & Fixing: The model processes a buggy code snippet and generates a corrected version.
5. Supported Languages: Works with multiple programming languages, making it versatile.
6. Fine-Tuning: Can be customized using additional labeled datasets for better performance.
7. Advantages: Provides state-of-the-art bug fixing, reduces manual debugging effort, and enhances coding efficiency.

Step 4: Develop AI Models

1. Model Selection:
   - Use CodeLlama, a pretrained LLM optimized for code understanding and generation.
   - Can be fine-tuned for better bug detection and fixing.
2. Model Components:
   - Bug Detection: Identify errors in code using the pretrained model.
   - Bug Classification: Categorize errors (syntax, logic, security, etc.).
   - Fix Generation: Suggest corrections for identified bugs.
3. Training Strategy:
   - Fine-tuning (optional): Train on labeled datasets of buggy and fixed code.
   - Transfer Learning: Utilize the existing knowledge of CodeLlama.
4. Evaluation Metrics:
   - Bug Detection: Accuracy, Precision, Recall, F1-score.
   - Fix Quality: BLEU Score, ROUGE Score.
5. Implementation:
   - Integrate the model into an interactive debugging system.
   - Ensure scalability for different programming languages.

Summary of Progress

Step 1: Defined Objectives & Requirements

Step 2: Set Up Development Environment

Step 3: Utilized a Pretrained Model (CodeLlama)

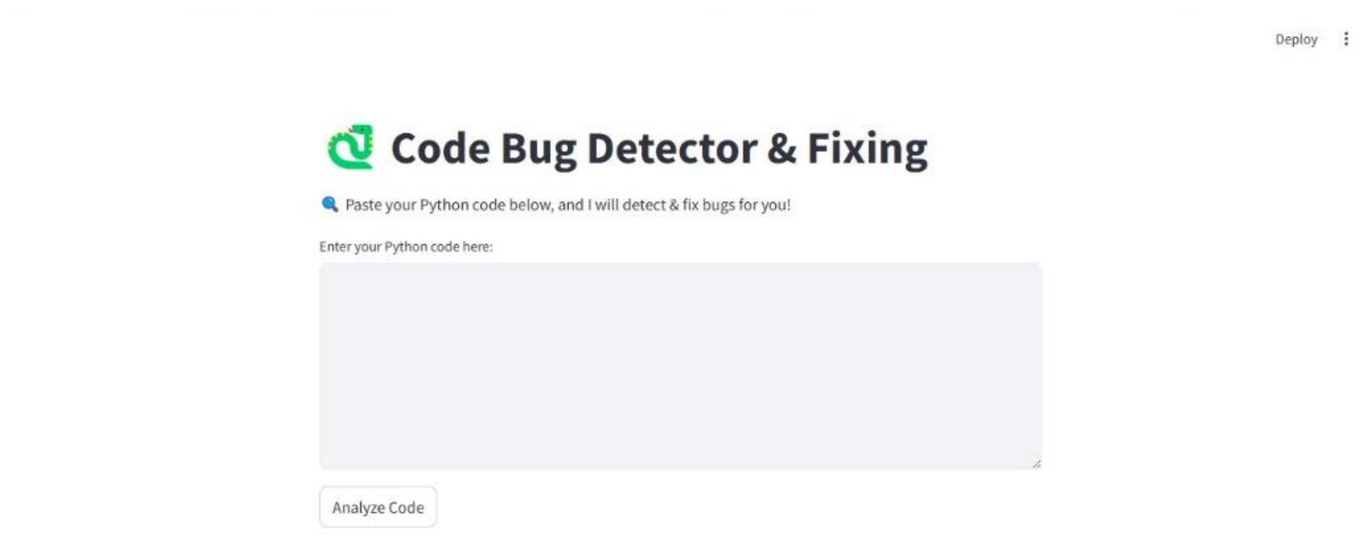
Step 4: Developed AI Models

Next Steps:

1. Fine-Tuning the Model – Train CodeLlama on buggy and corrected code to improve accuracy and adaptability.

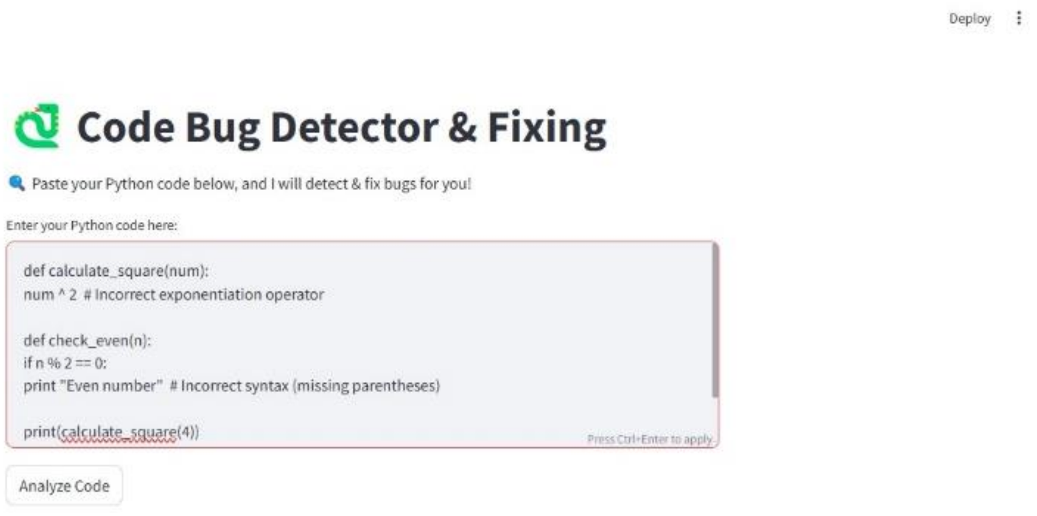
2. Model Evaluation – Assess performance using Accuracy, F1-score, BLEU, and ROUGE to refine predictions.
3. Integration & Deployment – Develop an API, IDE plugin, or web tool for real-time bug detection and fixing.
4. Performance Optimization – Enhance speed using GPU acceleration and efficient memory management.
5. User Testing & Feedback – Gather insights from real-world usage to improve accuracy and usability.

Result



🔍 Paste your Python code below, and I will detect & fix bugs for you!

Interface

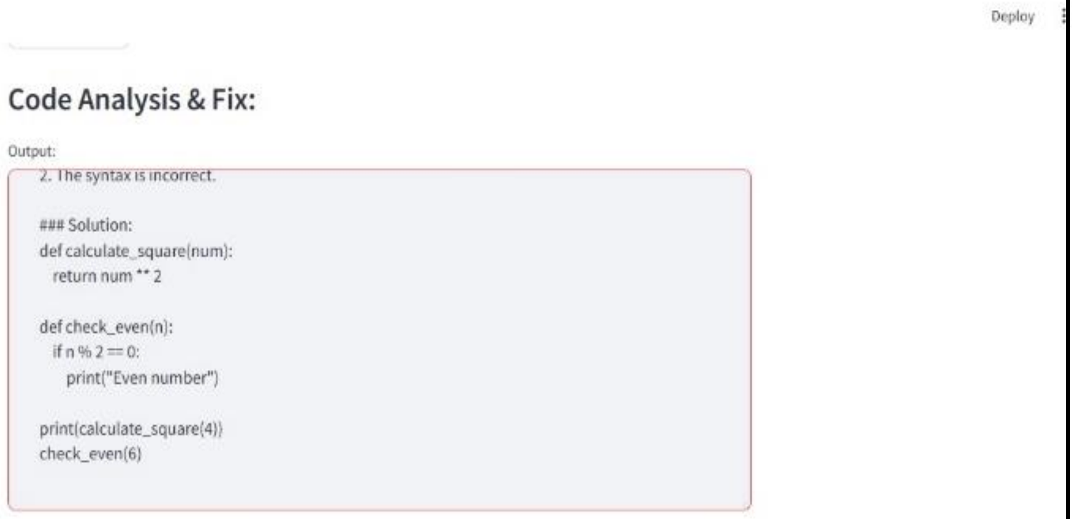

Deploy
Code Bug Detector & Fixing
🔍 Paste your Python code below, and I will detect & fix bugs for you!
Enter your Python code here:
Press Ctrl+Enter to apply
Analyze Code

```
def calculate_square(num):
num ^ 2  # Incorrect exponentiation operator

def check_even(n):
if n % 2 == 0:
print "Even number"  # Incorrect syntax (missing parentheses)

print(calculate_square(4))
```

Bug_Detection_And_Fixing_Input



```
2. The syntax is incorrect.

### Solution:
def calculate_square(num):
  return num ** 2

def check_even(n):
  if n % 2 == 0:
    print("Even number")

print(calculate_square(4))
check_even(6)
```

Bug_Detection_And_Fixing_Output

Conclusion

This report presents an AI-driven bug detection and fixing system using CodeLlama, showcasing how deep learning can enhance software debugging. The model successfully identifies errors and suggests fixes, automating a crucial part of software development. While the system performs well in detecting syntax and logical errors, future improvements can enhance its accuracy, adaptability, and integration into development environments. The project highlights the potential of AI in reducing debugging time, improving code quality, and assisting developers in writing more robust code efficiently.

The increasing complexity of software development necessitates more advanced debugging tools, and AI-based models like CodeLlama provide an effective solution. By automating bug detection and correction, developers can focus on optimizing code performance rather than spending extensive time debugging. Moreover, the continuous advancements in AI and natural language processing will further refine this technology, making it more reliable and widely applicable.

In conclusion, this AI-powered system represents a significant step toward the automation of software debugging. The combination of machine learning and programming analysis holds immense potential for the future of software development. With further improvements, such models could be seamlessly integrated into development pipelines, enhancing software quality and developer productivity worldwide.