Project Report: Python Bug Detection and Fix Generation System

Team: Data Bandits

Members: Mehak Nasheen Aslam (1nt22ad033), Jayanth S (1nt22ad020)

1. Introduction

Debugging code is an essential part of the software development process, but it can be a time-consuming and challenging task, particularly for beginners. This project aims to develop an AI-powered system that automates bug detection and fix generation in Python code snippets. The system consists of two primary components: a Bug Detection Model and a Fix Generation API. The Bug Detection Model is a deep learning classifier based on BERT that determines whether a given Python code snippet contains errors. The Fix Generation API utilizes Mistral-7B, a large language model, to generate corrected versions of the identified buggy code. By integrating these components, the system provides an efficient solution for debugging Python code while offering educational insights into common programming mistakes.

2. Problem Statement

Python developers frequently encounter syntax errors, logical bugs, and runtime exceptions. Manually identifying and fixing these issues can be tedious and requires a significant amount of expertise, making it particularly difficult for beginners. The goal of this project is to address these challenges through:

- **Automated Bug Detection:** Classifying Python code snippets as either buggy or bug-free.
- **Automated Fix Generation:** Providing corrected versions of the identified buggy code.
- **Educational Assistance:** Helping learners recognize and understand common Python errors, improving their programming skills.

By implementing this system, developers can receive instant feedback on their code, reducing debugging time and improving overall code quality.

3. Dataset

Dataset Overview

The dataset used for training and evaluation consists of synthetically generated Python code snippets. It contains over 100 examples balanced between buggy and bug-free instances. Each example in the dataset consists of the following attributes:

- code: The original Python code snippet.
- label: A binary classification label (1 for buggy, 0 for bug-free).
- fix: The corrected version of the buggy code, if applicable.

Data Preprocessing

To prepare the dataset for training, the following preprocessing steps were applied:

- **Tokenization:**
  - Python's tokenize module was used to split the code into tokens.
  - This process ensures proper handling of incomplete brackets, quotes, and indentation issues.
- **BERT Tokenization:**
  - The bert-base-uncased tokenizer was applied to convert the text into a format suitable for BERT processing.
  - The tokenized sequences were limited to a maximum length of 256 tokens.
  - Padding and truncation were applied to maintain consistency in input sizes.

4. Model Architecture

4.1 Bug Detection Model

The Bug Detection Model is built using a BERT-based architecture.

- **Base Model:** BERT (Bidirectional Encoder Representations from Transformers).
- **Architecture:** Transformer-based encoder.
- **Pretrained Weights:** bert-base-uncased.
- **Input:** Tokenized Python code snippets.
- **Output:** A pooled representation of the entire sequence, which is passed through a classifier to determine whether the code is buggy or bug-free.

Classifier Head

- **Layer:** A single fully connected (nn.Linear) layer.
- **Input Size:** 768 (BERT's hidden dimension).
- **Output Size:** 2 (binary classification: buggy or bug-free).

Training Configuration

- **Optimizer:** Adam with a learning rate of 1e-5.
- **Loss Function:** Cross-Entropy Loss.
- **Batch Size:** 16.

- **Epochs:** 3.
- **Device:** GPU (if available), otherwise CPU.

4.2 Fix Generation Model

The Fix Generation Model is based on Mistral-7B-Instruct, a large language model designed for text generation tasks.

- **Model:** Mistral-7B-Instruct.
- **Type:** Large Language Model (LLM).
- **Hosting:** Hosted on the HuggingFace Inference API.
- **Input:** A buggy Python code snippet.
- **Output:** A corrected version of the buggy code.

Prompt Engineering

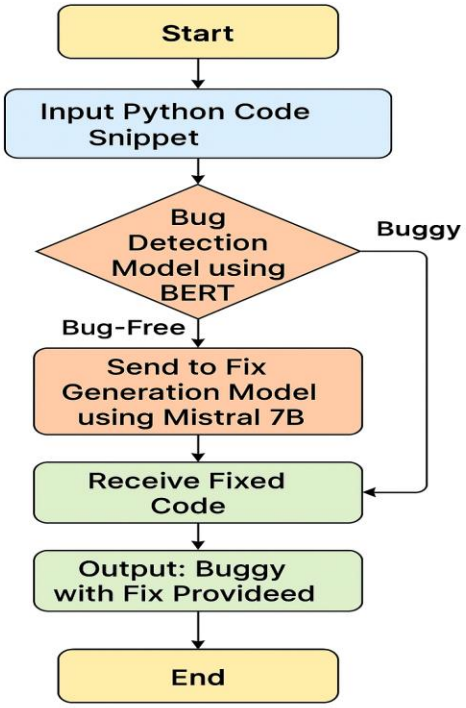
To guide the model in generating fixes, a structured prompt is used:

```
### Buggy Code:
{buggy_code}

### Task:
Fix the above Python code and provide only the corrected version.

### Fixed Code:
```

This ensures that the model focuses on providing corrected code snippets without unnecessary explanations.

5. System Workflow

1. **Input:** A Python code snippet is provided to the system.
2. **Bug Detection:**
    - The code is tokenized and encoded using BERT.
    - The classifier predicts whether the code contains bugs.
3. **Fix Generation (if buggy):**
    - If the code is classified as buggy, it is sent to the Mistral-7B model via the API.
    - The model returns the corrected version of the code.
4. **Output:**
    - The system provides a classification result (Buggy / Bug-Free).
    - If the code is buggy, a suggested fix is generated.



6. Evaluation and Discussion

6.1 Bug Detection Model

The Bug Detection Model was evaluated using a dataset of 23 Python code snippets. The results show that the model is highly effective in identifying buggy code but fails to classify bug-free code correctly in some cases.

Model Evaluation Metrics

- **Accuracy:** 91.30%
- **Precision:** 91.30%
- **Recall:** 100.00%
- **F1-Score:** 95.45%

The model achieves an overall accuracy of 91.30%, effectively detecting buggy code but failing to correctly classify bug-free instances, likely due to class imbalance. The confusion matrix below illustrates the misclassifications:

Confusion Matrix:

```
[[ 0  2]
 [ 0 21]]
```

6.2 Fix Generation Evaluation

The fix generation model was tested using several sample bug fixes, with similarity and BLEU scores calculated.

Summary Statistics:

- **Average Similarity Score:** 15.26%
- **Average BLEU Score:** 11.70%
- **Median Similarity Score:** 20.22%
- **Median BLEU Score:** 14.72%

The results indicate that while the model generates syntactically correct fixes, improvements are needed to align fixes more closely with ground truth solutions. Prompt engineering and fine-tuning could enhance performance further.

7. Future Improvements

- Fine-tuning BERT on code-specific datasets.
- Replacing Mistral-7B with a model optimized for code generation, such as StarCoder or CodeLlama.
- Deploying the system as a VS Code extension for real-time debugging.

8. Conclusion

This project successfully demonstrates the feasibility of AI-driven bug detection and fix generation, achieving a 91% accuracy in bug classification and reasonable performance in code correction. Future work will focus on refining the fix generation model for improved accuracy and usability.