THE HAWKS

BUG DETECTION AND FIXING

ABSTRACT

Bug detection and automatic code correction are essential for improving software reliability. This project presents a Streamlit-based Buggy Code Detector and Fixer, leveraging LSTM-based deep learning for bug detection and a Hugging Face API for code correction. The model is trained on a labeled dataset of source code, using tokenization and sequence padding for preprocessing. To address class imbalance, Random Oversampling is applied. A Bidirectional LSTM network is used to classify code as buggy or bug-free, achieving balanced accuracy through class weighting. The trained model is deployed using Streamlit, allowing users to input source code and receive real-time feedback. If a bug is detected, the system invokes a pretrained Hugging Face model to generate a corrected version of the code. This tool enhances developer productivity by automating bug identification and resolution.

INTRODUCTION

Software development often involves debugging, a time-consuming and complex process that requires identifying and fixing errors in the source code. Traditional debugging methods rely on manual inspection and static analysis tools, which can be inefficient, especially for large-scale projects. To address this challenge, machine learning-based bug detection and automated code correction have gained significant attention.

This project presents a Streamlit-based Buggy Code Detector and Fixer, integrating deep learning and natural language processing (NLP) techniques to classify source code as buggy or bug-free. A Bidirectional LSTM (Long Short-Term Memory) model is trained on labeled code samples to identify potential bugs. Additionally, a Hugging Face API is utilized to automatically suggest fixes for buggy code, enhancing software quality and reducing debugging time.

The system follows a structured workflow, including data preprocessing, model training, bug detection, and automated correction. The Tokenization and sequence padding techniques transform source code into numerical sequences, enabling the LSTM model to learn patterns in the data.

Class imbalance is handled using Random Oversampling, ensuring that the model generalizes well to real-world datasets. The final deployment uses Streamlit, a lightweight Python framework, allowing users to input code, check for bugs, and receive AI-generated fixes in real-time. By integrating deep learning with practical deployment, this tool serves as an intelligent assistant for developers, streamlining the debugging process and improving code reliability.

METHODOLOGY

The proposed Buggy Code Detector and Fixer follows a systematic approach, integrating deep learning for bug detection and a Hugging Face API for automatic code correction. The methodology consists of several key stages:

Data Acquisition and Preprocessing

The dataset consists of labeled source code samples categorized as "Bug-Free" (0) or "Buggy" (1). The dataset is first loaded using Pandas, and any missing values are handled to ensure data integrity. To prepare the data for the deep learning model, tokenization is performed using the Keras Tokenizer, converting source code into sequences of integers. An Out-of-Vocabulary (OOV) token (<OOV>) is included to handle unseen words. Since sequence lengths vary, padding is applied using pad_sequences() to ensure uniform input size (max_length). Class imbalance is addressed using Random Oversampling (ROS) from the imblearn library, ensuring an equal distribution of buggy and bug-free samples.

Model Development

A Bidirectional Long Short-Term Memory (Bi-LSTM) network is implemented for binary classification. The model begins with an Embedding layer, which converts tokenized sequences into dense vector representations. Two Bidirectional LSTM layers (128 and 64 units each) capture sequential dependencies in both forward and backward directions. To prevent overfitting, Dropout layers with a dropout rate of 0.5 are added. A fully connected dense layer (64 neurons, ReLU activation) with L2 regularization refines feature extraction. Finally, the output layer, a single neuron with a sigmoid activation function, classifies the input as buggy or bug-free. The model is compiled using Binary Cross-Entropy as the loss function and Adam optimizer (learning rate = 0.0005).

To handle any residual class imbalance, class weights are computed and applied during training. The model is trained for 10 epochs with a batch size of 32, using an 80%-20% training-validation split.

Model Evaluation and Saving:

Once trained, the model is evaluated on the validation dataset, measuring accuracy and loss to assess performance. The final trained model is then saved in HDF5 format (.h5) for deployment. Additionally, the Tokenizer and max_length value are stored using Pickle, ensuring consistent preprocessing during inference.

Deployment using Streamlit:

The trained LSTM model is integrated into a Streamlit-based web application, allowing users to input Python code for analysis. The user interface includes a text area for input, a "Check Code" button for classification, and visual feedback on whether the code is buggy or bug-free. If the model predicts the presence of a bug, an AI-generated fix is suggested.

Automatic Code Fixing via Hugging Face API

For buggy code, a Hugging Face Transformer model is used to generate a corrected version. The input code is sent to the Hugging Face API via an HTTP POST request using requests.post(), where the API processes the input and returns a fixed version of the code. The response is parsed, and the corrected code is displayed within the Streamlit app.

User Interaction and Final Output

Users receive instant feedback on whether their code contains bugs or not. If a bug is detected, the AI-generated fixed code is displayed using syntax highlighting (st.code()), improving readability. By integrating deep learning, NLP, and automated code correction, this tool enhances debugging efficiency and streamlines the software development process.

ISSUES FACED AND RESULTS

During the development of the Buggy Code Detector and Fixer, several challenges were encountered, including class imbalance, where bug-free samples outnumbered buggy ones, leading to biased predictions. This was resolved using Random Oversampling (ROS) to balance the dataset. Handling variable-length code snippets was another issue, addressed by carefully selecting an optimal padding length (max_length) to retain meaningful information without increasing computational complexity. The LSTM model initially suffered from overfitting, which was mitigated using dropout layers, L2 regularization, and class weighting to enhance generalization. Integrating the Hugging Face API for code correction also posed difficulties, as some responses were inconsistent or incomplete, requiring fine-tuning API parameters and implementing exception handling. Deployment on Streamlit introduced performance concerns, which were resolved by preloading the tokenizer and trained model to reduce latency. Despite these challenges, the system performed effectively, achieving an 85-90% validation accuracy, successfully distinguishing between buggy and bug-free code. The real-time Streamlit application allowed users to instantly analyze their code and receive AI-generated fixes, significantly reducing manual debugging efforts and improving developer productivity. This project demonstrates the potential of deep learning and NLP in automating software debugging, making the process faster and more efficient.

UNIQUENESS OF THE PROJECT

This project stands out due to its end-to-end integration of bug detection and automated fixing within a single, user-friendly platform. Unlike traditional static analysis tools, it uses a deep learning model (Bidirectional LSTM) to learn patterns in source code and detect bugs contextually. Moreover, it leverages the Hugging Face StarCoder API to generate human-like code fixes, enabling intelligent code repair. The use of Streamlit for real-time interaction makes it accessible even to non-expert users, while oversampling techniques ensure balanced learning from imbalanced data. This combination of classification and generation in a deployed app offers a novel, AI-powered approach to improving code quality automatically.

CONCLUSION

The development of the Buggy Code Detector and Fixer demonstrates the effectiveness of deep learning and NLP in automating software debugging. By leveraging a Bidirectional LSTM model, the system successfully classifies source code as buggy or bug-free, achieving a high validation accuracy of 85-90%. The integration of the Hugging Face API further enhances the tool by providing automated bug fixes, reducing manual debugging efforts. The Streamlit-based interface ensures real-time interaction, making it user-friendly and accessible for developers. Despite challenges such as class imbalance, overfitting, and API response inconsistencies, strategic solutions like oversampling, dropout regularization, and response handling significantly improved the model's performance and reliability. Overall, this project highlights the potential of AI-powered debugging tools in improving software quality and developer efficiency. Future enhancements could include support for multiple programming languages, improved bug-fix suggestions.