Bug Detection and Fixing using Streamlit Dashboard

# Introduction

In the ever-evolving field of software engineering, identifying and resolving bugs plays a critical role in ensuring robust and error-free applications. Manual debugging, however, can be tedious and prone to oversight. This project presents a machine learning-powered system that automates bug detection and suggests potential fixes. The results are showcased through a user-friendly Streamlit dashboard, enabling developers to easily interact with the system.

# Motivation

With the increasing size and complexity of codebases, conventional methods of manual debugging have become inefficient. There is a growing need for intelligent systems that can automate bug detection and suggest resolutions. Machine learning offers a promising solution by learning patterns from historical bug data and applying them to new code. By integrating this capability with an intuitive dashboard, the project aims to provide developers with a powerful tool to identify and resolve bugs faster.

# Problem Definition

The primary objective is to develop a system that can:  
- Automatically detect bugs in the source code.  
- Classify the type or severity of bugs using machine learning.  
- Recommend potential fixes based on learned patterns.  
- Provide an interactive dashboard for users to input code and view results in real-time.

# Prerequisite Knowledge

To understand and contribute to this project, one should be familiar with:  
- Python programming.  
- Basics of Machine Learning and Natural Language Processing (NLP).  
- Streamlit for web-based dashboards.  
- Git for version control.  
- Software development lifecycle and debugging practices.

# External Interface Requirements

- Input Interface: Users can input code directly through a text box or upload a code file.  
- Database/API Interface: Optional integration with bug databases (e.g., GitHub Issues, Bugzilla).  
- Output Interface: The system returns analysis and suggestions in a visual format via Streamlit.

# User Interfaces

The front-end of the system is built using Streamlit, which includes:  
- Code input section (text or file upload).  
- Bug detection results (highlighted code, summary table).  
- Fix suggestions with explanations.  
- Visual analytics: bug frequency, severity charts, etc.  
- Model performance metrics (accuracy, F1-score, confusion matrix).

# Hardware Interfaces

- Processor: Intel i5 or above.  
- Memory: Minimum 8 GB RAM.  
- Storage: 500 GB HDD or SSD.  
- Optional: NVIDIA GPU for training deep learning models.

# Software Interfaces

- Operating System: Windows/Linux/Mac.  
- Python 3.10+  
- Libraries: Streamlit, Scikit-learn, TensorFlow/PyTorch, Pandas, NumPy.  
- Jupyter Notebook for development and model training.  
- GitHub for code collaboration and versioning.

# Communication Interfaces

- Local web server via Streamlit (localhost or network IP).  
- Optional REST APIs to connect external bug tracking tools or IDEs.

# Project Design

The design follows a modular structure:  
1. Preprocessing Module: Tokenizes and analyzes source code.  
2. Bug Detection Module: Applies trained ML models to identify bugs.  
3. Fix Recommendation Module: Uses heuristics and ML to suggest code changes.  
4. Dashboard Module: Visualizes results in Streamlit.  
Each module is loosely coupled, allowing for scalability and upgrades.

# Project Architecture

The architecture is as follows:  
User Input → Preprocessing → ML Model → Fix Recommendation → Visualization (Streamlit).  
The model uses a labeled dataset of buggy and fixed code to learn patterns and classify new code segments.

# Project Specification

- Languages Supported: Currently Python (extendable).  
- Model Type: Classifier (e.g., Random Forest, LSTM, BERT for code).  
- Input Types: Text code, uploaded files.  
- Output Types: Bug classification, fix suggestions, visual metrics.

# Other Specification

- Modular and open-source architecture.  
- Pretrained models used for prototype demonstration.  
- Ability to adapt for continuous learning and feedback from users.

- Speeds up the debugging process.

-Helps identify critical bugs early in development.

-Enhance developer productivity.

-Can be integrated into existing development workflows

# Advantages

- Speeds up the debugging process.  
- Helps identify critical bugs early in development.  
- Enhances developer productivity.  
- Can be integrated into existing development workflows.

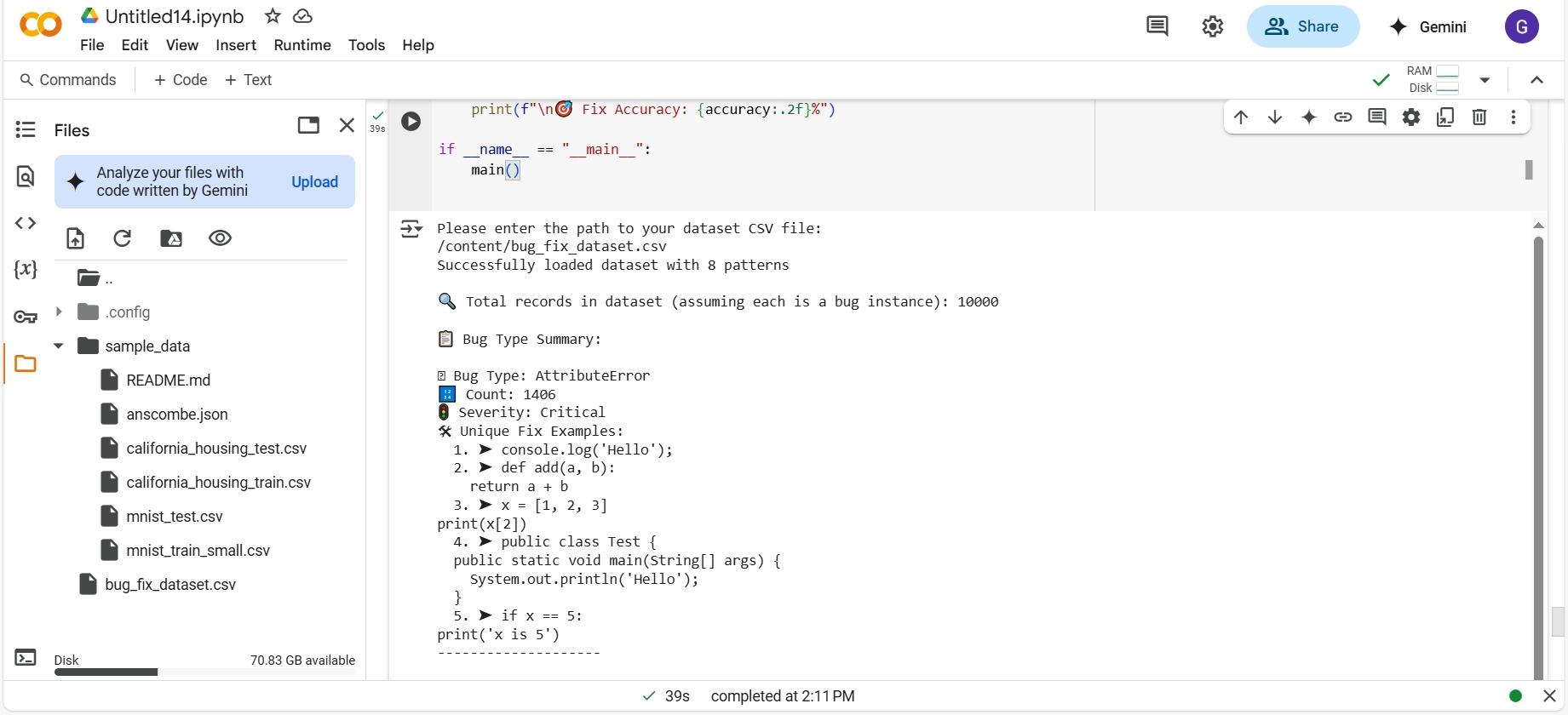
# Limitations

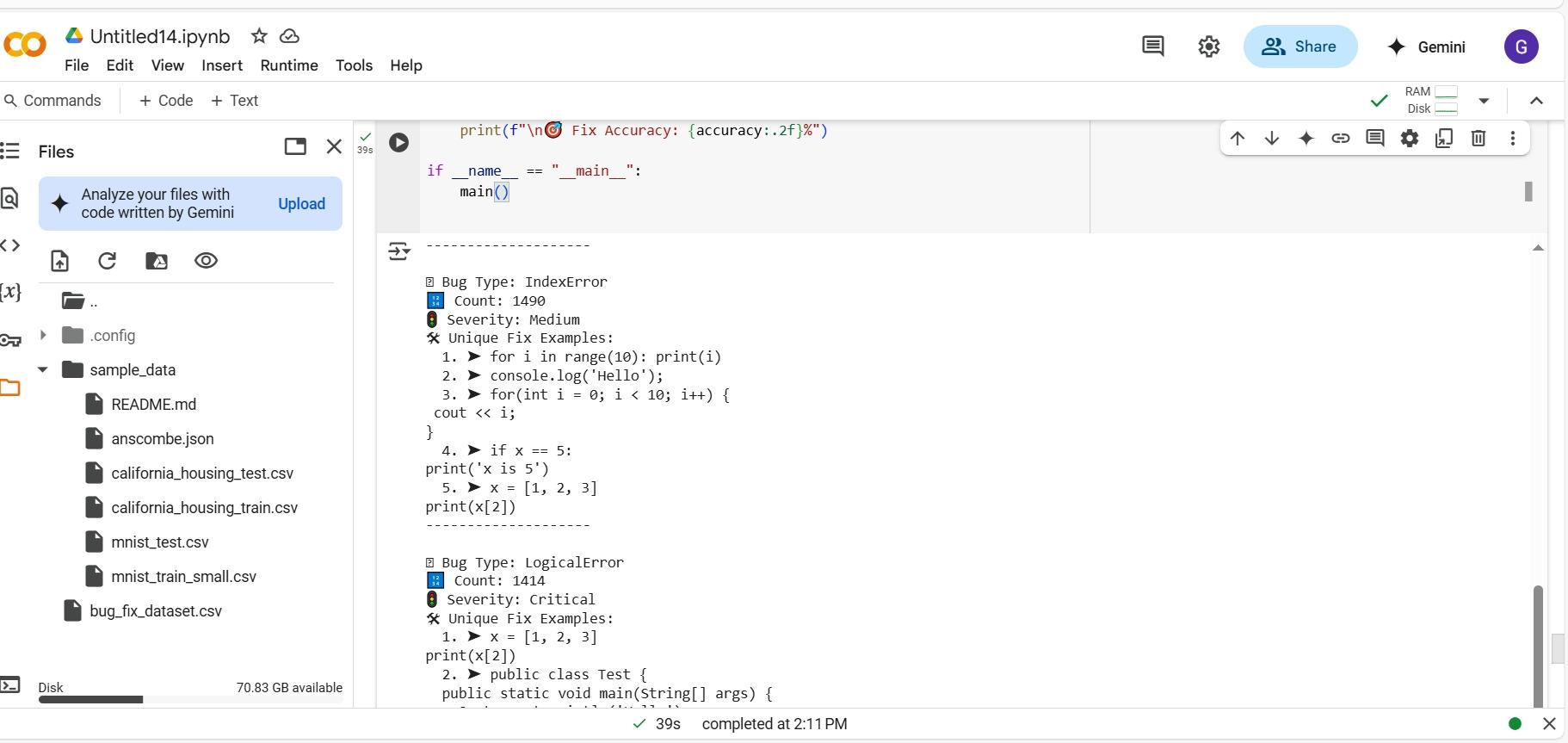
- Relies on quality and quantity of training data.  
- May not detect logical or context-specific bugs.  
- Fix suggestions may not always be applicable or optimal.  
- Currently supports only Python; extension required for other languages.

# Applications

- Educational platforms for teaching programming and debugging.  
- Code review automation in software teams.  
- Integration into IDEs and CI/CD pipelines.  
- Bug analytics and reporting tools.

# Code And Output Screenshots





# 

# 

# 

# Conclusions and Future Scope

This project successfully demonstrates how machine learning can be applied to automate the bug detection and fixing process. With the integration of an interactive dashboard, it becomes a practical tool for both learners and professionals. Though limited in scope currently, the system has the potential to evolve into a comprehensive bug management suite with multi-language support and deeper learning capabilities.

Integrating advanced NLP models for better bug understanding.

Expanding support to multiple programming languages.

Automating code fix suggestions using deep learning techniques.

Real-time collaboration features in the dashboard.

# Outcomes

- Developed a functional ML-based bug detection system.  
- Designed a responsive dashboard using Streamlit.  
- Gained hands-on experience in full-stack development and machine learning workflows.  
- Learned how to work collaboratively on real-world problems under the guidance of Intel Unnati.