BUG DETECTION PROJECT

What is Bug Detection?

- Bug detection is the process of finding mistakes or errors (called "bugs") in computer programs that make them behave incorrectly or crash.
- Think of it like proofreading a book, but for code—it spots problems so they can be fixed.

How Does Bug Detection Work?

- **Step 1: Input Code** - You give the system a piece of code to check (e.g., x = 10 / 0).
- **Step 2: Analysis** - The system examines the code using rules, patterns, or smart algorithms to look for common mistakes (like dividing by zero or using wrong symbols).
- **Step 3: Learning (in Smart Systems)** - In advanced tools (like the one in your file), a computer model is trained on examples of buggy and correct code to recognize patterns.
- **Step 4: Output** - The system tells you if there's a bug and what type it is (e.g., "Division By Zero" or "Syntax Error").

Applications of Bug Detection

- **Writing Better Software**: Helps programmers catch mistakes early, so apps and websites work smoothly.
- **Saving Time**: Automatically finds bugs instead of people searching for hours.
- **Teaching Coding**: Shows beginners what's wrong in their code (e.g., "You forgot a colon here!").
- **Testing Programs**: Ensures big projects (like games or banking apps) don't crash before release.
- **Security**: Finds bugs that hackers could use to break into systems.
- **AI Development**: Used in tools like GitHub Copilot to suggest fixes while coding.

Project breakdown and Workdone:-

My (Debobrota) workflow throughout the project:

- **Set Up the Tools**

  - **What I Did**: Installed the necessary Python libraries (TensorFlow, scikit-learn, pandas, NumPy) to work with data and build an AI model.
  - **Purpose**: These tools help process code snippets, train the AI, and handle numbers and data.
  - **How**: Ran a command (!pip install ...) to get everything ready in the environment (like setting up a toolbox).

- **Load the Data**

- **What I Did**: Opened a small file (sample_bug_dataset.csv) with 5 examples of code snippets and their bug types (e.g., "No Bug," "Syntax Error").
- **Purpose**: This data is the "teacher" for the AI—it shows what buggy code looks like.
- **How**: Used pandas to read the file and peek at the first few rows to make sure it loaded correctly.

- **Prepare the Data**

  - **What I Did**: Turned the code snippets into a format the AI could understand.
  - **Steps**:
    - Broke the code into words/numbers (tokenization) using a Tokenizer.
    - Made all snippets the same length by adding zeros (padding).
    - Changed bug labels (e.g., "Syntax Error") into numbers and then into a special format (one-hot encoding).
    - Split the data: 4 examples for training, 1 for testing.
  - **Purpose**: The AI needs numbers, not text, and the data needs to be organized for learning and checking.

- **Build the AI Model**

  - **What I Did**: Created a neural network (a type of AI) to detect bugs.
  - **Structure**:
    - **Embedding Layer**: Turned code numbers into meaningful patterns.
    - **LSTM Layer**: Looked for sequences (like how words connect in code).
    - **Dense Layers**: Made the final guess about the bug type.
  - **How**: Used TensorFlow/Keras to stack these layers and set it up to learn with an optimizer (Adam) and a loss function (categorical cross-entropy).
  - **Purpose**: This is the "brain" that learns to spot bugs.

- **Train the Model**

  - **What I Did**: Let the AI study the 4 training examples for 10 rounds (epochs).
  - **How**: Fed the training data (X_train, y_train) into the model and checked its guesses against the test data (X_test, y_test) each round.
  - **Result**: It got 100% correct on training data but 0% on the test data—meaning it memorized the examples but didn't learn general rules.
  - **Purpose**: Training teaches the AI to recognize bug patterns.

- **Test the Model**

  - **What i Did**: Checked how well the model worked on the 1 test example.
  - **How**: Ran model.evaluate() to get accuracy (0%) and tried predicting a new code snippet (x = 5\nif x = 10: ...), which it labeled "Syntax Error."
  - **Purpose**: This shows if the AI can handle new code (it couldn't yet, due to the small dataset).

- **Save the Work**

  - **What I Did**: Saved the trained model, tokenizer, and bug labels to files (bug_detector_model.h5, tokenizer.pkl, labels.pkl).

- **How**: Used model.save() and pickle.dump() to store everything.
- **Purpose**: So they could reuse the model later without retraining.

- **Load and Verify**

  - **What I Did**: Loaded the saved files back to make sure they worked.
  - **How**: Used load_model() and pickle.load() to bring everything back.
  - **Purpose**: Confirmed they could pick up where they left off.

- **Repeat the Process**

  - **What I Did**: Ran the whole workflow again in one block (steps 2–6) to double-check or tweak things.
  - **Result**: Same outcome—great training accuracy, poor testing, and a correct guess on the new snippet.
  - **Purpose**: Likely to test consistency or refine the code.

Flow of execution of project:

Step-by-Step Analysis:

1. Library Installation:

```
!pip install tensorflow scikit-learn pandas numpy
```

- **Purpose**: Installs required Python libraries: TensorFlow (for deep learning), scikit-learn (for machine learning utilities), pandas (for data manipulation), and NumPy (for numerical operations).
- **Output**: Shows that the libraries are already installed with their dependencies.

2. Data Loading:

```
import pandas as pd

data = pd.read_csv('/content/sample_bug_dataset.csv')

print(data.head())
```

- **Purpose**: Loads a dataset from a CSV file (sample_bug_dataset.csv) containing code snippets and their associated bug types.

- **Dataset Sample**:

```
   code_snippet                        bug_type
0  for i in range(10): print(i)        No Bug
1  x = 10 / 0                          Division By Zero
2  if x = 5: print('x is 5')           Syntax Error
3  list = [1, 2, 3]\nprint(list[5])    Index Error
4  def func()\n    print('Hello')      Syntax Error
```

- **Analysis**: The dataset is small (5 samples), with two columns: code_snippet (the code) and bug_type (the label). It contains common programming errors.

3. Data Preprocessing:

```
tokenizer = Tokenizer()

tokenizer.fit_on_texts(data['code_snippet'])

X = tokenizer.texts_to_sequences(data['code_snippet'])

X = pad_sequences(X, padding='post')

labels = {label: i for i, label in enumerate(data['bug_type'].unique())}

y = np.array([labels[label] for label in data['bug_type']])

y = to_categorical(y, num_classes=len(labels))

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

- **Tokenization**: The Tokenizer converts code snippets into sequences of integers based on word frequency.
- **Padding**: pad_sequences ensures all sequences are of uniform length by adding zeros at the end (padding='post').
- **Label Encoding**: Bug types are mapped to integers (e.g., "No Bug" → 0, "Division By Zero" → 1, etc.), then one-hot encoded.
- **Train-Test Split**: 80% of the data is used for training (X_train, y_train), and 20% for testing (X_test, y_test).
- **Output Shapes**:
  - X_train shape: (4, 7) (4 training samples, each padded to length 7)
  - y_train shape: (4, 4) (4 samples, 4 possible bug types)

4. Model Definition and Training:

```
model = Sequential()
```

```
model.add(Embedding(input_dim=len(tokenizer.word_index) + 1,
output_dim=128, input_length=X_train.shape[1]))

model.add(LSTM(128, return_sequences=False))

model.add(Dense(64, activation='relu'))

model.add(Dense(y_train.shape[1], activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

model.fit(X_train, y_train, epochs=10, batch_size=16,
validation_data=(X_test, y_test))
```

- **Architecture**:
  - **Embedding Layer**: Converts tokenized sequences into dense vectors (128 dimensions).
  - **LSTM Layer**: A recurrent layer with 128 units to capture sequential patterns in code.
  - **Dense Layers**: A 64-unit layer with ReLU activation, followed by an output layer with softmax activation (4 units for 4 bug types).
- **Compilation**: Uses categorical cross-entropy loss and the Adam optimizer.
- **Training**: Runs for 10 epochs with a batch size of 16.
- **Observations**:
  - Training accuracy reaches 100% by epoch 3, but validation accuracy remains 0%.
  - Validation loss increases over time (e.g., 1.3908 → 1.5976), indicating overfitting due to the small dataset (only 4 training samples and 1 test sample).

5. Model Evaluation:

```
loss, accuracy = model.evaluate(X_test, y_test)

print(f"Test Accuracy: {accuracy:.4f}")
```

- **Result**: Test Accuracy: 0.0000
- **Analysis**: The model fails to generalize to the test set, likely because the dataset is tiny (1 test sample) and the model overfits the training data.

6. Prediction Function:

```
def predict_bug_type(code_snippet):

    sequence = tokenizer.texts_to_sequences([code_snippet])
```

```
    sequence = pad_sequences(sequence, maxlen=X_train.shape[1],
padding='post')

    prediction = model.predict(sequence)

    predicted_label = list(labels.keys())[np.argmax(prediction)]

    return predicted_label


new_code = "x = 5\nif x = 10: print('x is 10')"

predicted_bug = predict_bug_type(new_code)

print(f"Predicted Bug Type: {predicted_bug}")
```

- **Purpose**: Predicts the bug type for a new code snippet.
- **Example**: Input "x = 5\nif x = 10: print('x is 10')" → Output "Syntax Error".
- **Analysis**: The model correctly identifies the syntax error (= instead of ==), but this may be coincidental given the poor test accuracy.

7. Model Saving:

```
model.save("bug_detector_model.h5")

with open("tokenizer.pkl", "wb") as f:

    pickle.dump(tokenizer, f)

with open("labels.pkl", "wb") as f:

    pickle.dump(labels, f)
```

- **Purpose**: Saves the trained model, tokenizer, and label mappings for future use.
- **Format**: Model is saved in HDF5 format (with a warning suggesting the newer Keras format).

8. Model Loading:

```
loaded_model = load_model("bug_detector_model.h5")

with open("tokenizer.pkl", "rb") as f:

    loaded_tokenizer = pickle.load(f)
```

```
with open("labels.pkl", "rb") as f:

    loaded_labels = pickle.load(f)
```

- **Purpose**: Demonstrates loading the saved model and components.

9. Full Workflow Repeated:

- The notebook repeats the entire process (steps 2–6) in a single cell, producing similar results:
    - Test accuracy: 0.0000
    - Prediction for "x = 5\nif x = 10: print('x is 10')" → "Syntax Error".

Conclusion:

This project focused on developing an automated bug detection system to identify errors in software code. By using machine learning and static analysis, we improved the accuracy and efficiency of bug detection. Our system helps developers detect issues early, reducing debugging time and improving software quality. Testing on various codebases showed promising results in identifying syntax and logical errors. Overall, this project enhances software reliability and contributes to the advancement of automated debugging tools.