Replication Instructions

This document provides detailed instructions for replicating the comparative results of bug report classification models on performance-related bug detection in deep learning frameworks.

Environment Setup

1. Clone the repository (two options):

   Using HTTPS:

   ```
   git clone https://git.cs.bham.ac.uk/ceg212/Caleb_ISE_Coursework.git
   ```

   Using SSH:

   ```
   git clone git@git.cs.bham.ac.uk:ceg212/Caleb_ISE_Coursework.git
   ```

2. Navigate to the project directory:

   ```
   cd Caleb_ISE_Coursework
   ```

3. Set up your environment (two options):

   Option A: Using a virtual environment

   ```
   python -m venv venv
   source venv/bin/activate  # On Windows: venv\Scripts\activate
   pip install -r requirements.txt
   ```

   Option B: Using your IDE or global environment Simply install the dependencies directly:

   ```
   pip install -r requirements.txt
   ```

4. Download required NLTK resources:

   ```
   python download_nltk_resources.py
   ```

5. Download the spaCy model:

   ```
   python -m spacy download en_core_web_sm
   ```

Dataset Preparation

The project includes code to generate synthetic test datasets if the actual datasets are not available:

1. Create sample datasets for testing:

   ```
   cd lab1
   python create_sample_dataset.py
   ```

   This will create synthetic dataset files (e.g., `tensorflow.csv`) in the `datasets` directory.

2. To use real-world datasets, place them in the `datasets` directory with framework names as filenames:

   - `tensorflow.csv`

    - `pytorch.csv`
    - `keras.csv`
    - `incubatormxnet.csv`
    - `caffe.csv`

3. Ensure datasets have the following columns:

    - `Title`: The bug report title
    - `Body`: The bug report description/body
    - `class`: Binary label (1 for performance bug, 0 for non-performance bug)

Running the Experiments

Below are different ways to run experiments, from quick single-framework tests to comprehensive evaluations. Make sure you are in the lab1 folder when running these experiments (cd lab1).

1. Quick Testing on a Single Framework

To quickly test models on just one framework (e.g., TensorFlow):

```
# Test hybrid model vs baseline on TensorFlow
python test_hybrid_model.py --framework tensorflow

# Test intermediate model vs baseline on TensorFlow
python test_intermediate_model.py --framework tensorflow

# Test all models on TensorFlow
python test_all_models.py --framework tensorflow
```

You can replace `tensorflow` with any available framework: `pytorch`, `keras`, `caffe`

2. Testing on Specific Frameworks

To test on multiple, specific frameworks:

```
# Test hybrid model on TensorFlow and PyTorch
python test_hybrid_model.py --frameworks tensorflow pytorch

# Test intermediate model on Keras and Caffe
python test_intermediate_model.py --frameworks keras caffe
```

3. Testing on All Frameworks

To compare one model against the baseline across all frameworks:

```
# Compare hybrid model to baseline on all frameworks
python test_hybrid_model.py

# Compare intermediate model to baseline on all frameworks
python test_intermediate_model.py

# Test all models on all frameworks for comprehensive summary and plots in
        comprehensive_results folder (Note: this will take a longer time), but replicates
        the results shown in my report.
python test_all_models.py
```

4. Comprehensive Evaluation

For the most comprehensive evaluation (runs all models on all frameworks with multiple runs for statistical validity):

```
python evaluation_framework.py
```

This command generates detailed statistics, visualisations, and a summary report.

5. Faster Comprehensive Evaluation

If the full evaluation is too time-consuming, you can reduce the scope:

```
# Reduce sample size and number of runs
python evaluation_framework.py --sample_ratio 0.2 --n_runs 2

# Run only on specific frameworks
python evaluation_framework.py --frameworks tensorflow pytorch
```

Output Files

After running experiments, examine the following output:

1. Results Directory (`results/`)

Contains raw data from experiment runs: - `*_multiple_runs.csv` - Results from multiple evaluation runs - `evaluation_summary.md` - Overall performance summary

2. Plots Directory (`results/plots/`)

Contains visualisations: - `*_f1_score_comparison.png` - Bar charts comparing F1 scores - `*_precision_recall.png` - Precision and recall comparisons - `*_f1_score_boxplot.png` - Statistical distribution of F1 scores - `*_training_time.png` - Training time comparisons

3. Comprehensive Results (`results/comprehensive_results/`)

Contains detailed outputs for in-depth analysis: - Framework-specific performance tables - Cross-framework comparisons - Detailed metric breakdowns - Statistical test results

Interpreting Results

Evaluation Summary File

The `evaluation_summary.md` file shows performance metrics for each model across frameworks: - **Precision**: Higher values mean fewer false positives (incorrect performance bug identifications) - **Recall**: Higher values mean fewer false negatives (missed performance bugs) - **F1 Score**: Balance between precision and recall - **Training/Prediction Time**: Resource requirements

Visualisation Files

1. **F1 Score Comparison**:
   - Bar charts comparing overall performance
   - The hybrid model should consistently show higher bars
   - If a model is missing, it may have failed due to memory constraints
2. **Precision-Recall Comparison**:
   - Shows trade-offs between precision and recall
   - The hybrid model typically shows better balance
3. **F1 Score Boxplots**:
   - Shows statistical distribution across multiple runs
   - Narrower boxes indicate more consistent performance
   - Higher boxes indicate better performance

Expected Results

After running the experiments, you should see results similar to these:

Framework: TensorFlow

Model	Precision	Recall	F1 Score
Baseline	~0.358	~0.939	~0.518
Intermediate	~0.287	~0.818	~0.424
Hybrid	~0.509	~0.838	~0.633

Framework: PyTorch

Model	Precision	Recall	F1 Score
Baseline	~0.239	~0.788	~0.366
Intermediate	~0.175	~0.848	~0.286
Hybrid	~0.422	~0.697	~0.522

Overall Performance (All Frameworks)

Model	Precision	Recall	F1 Score
Baseline	~0.353	~0.862	~0.496
Intermediate	~0.238	~0.917	~0.374
Hybrid	~0.467	~0.770	~0.576

Note: Your exact values may vary slightly due to: - Random initialisation in the models - Different sampling in train/test splits - Hardware performance variations

My hybrid model should consistently outperform the baseline and intermediate models in F1 score, with typical improvements of 14-20% over the baseline across frameworks.

Troubleshooting

- **Memory Issues**: If you encounter memory errors:

```
python evaluation_framework.py --sample_ratio 0.1 --n_runs 1
```

- **Speed Issues**: To speed up evaluation:

  ```
  # Reduce feature dimensions
  python test_hybrid_model.py --framework tensorflow --max_features 1000
  ```

- **Missing Models in Results**:
  - The hybrid model may be absent from some visualisations if memory constraints prevented completion
  - Only models that complete the evaluation process appear in the results

- **Missing NLTK Data**:

  ```
  python download_nltk_resources.py
  ```

- **SpaCy Model Error**:

  ```
  python -m spacy download en_core_web_sm
  ```

If issues persist, check the error messages or try testing on a smaller dataset.