CodeReviewer.AI

Final PPT

Team members:

Yogev Cohen

Dudi Ohayon

Romy Somkin

Introduction

Goal & Motivation

- **Build an automated code review model** to classify code changes as high or low quality.
- **Transfer code review knowledge** from Java to C++, even when C++ data is limited.

Why Does This Matter?

- Manual code review is slow and error-prone.
- Many languages (like C++) lack enough labeled review data.
- **Data augmentation and transfer learning** can make automated code review more universal and practical.

Our Approach

- **Leverage Java data** and LLMs to generate synthetic C++ review data.
- **Train and compare** models on real vs. synthetic C++ data to test the effectiveness of language-agnostic augmentation.

Formal Task Specification

Input

- Code change:
  - Composed of the original code (oldf) and the patch (patch)

output

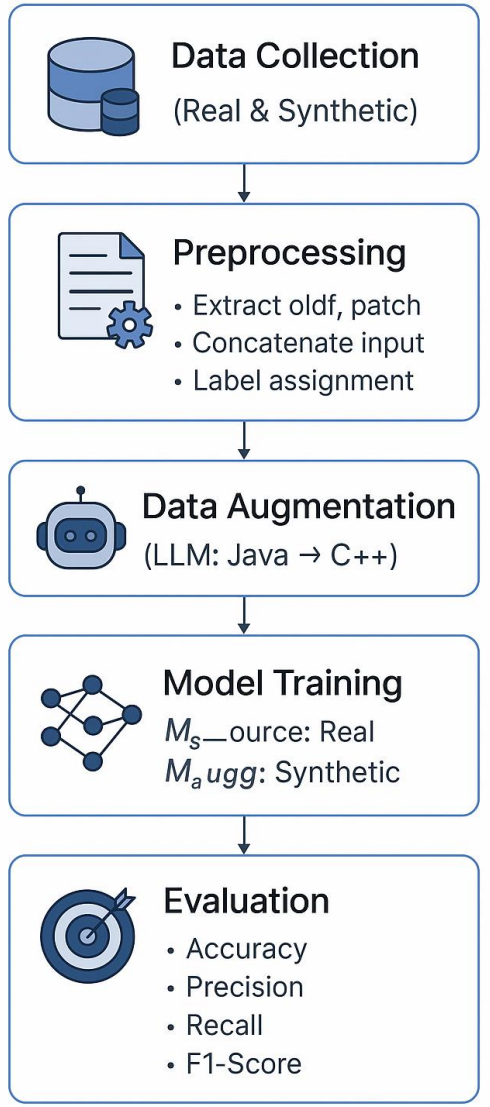
- Binary label
  - 0- High quality, no further review needed
  - 1- Low quality, requires further review.

metrics

- **Accuracy:** Proportion of correct predictions
- **Precision:** True positives among predicted positives
- **Recall:** True positives among actual positives
- **F1-score:** Harmonic mean of precision and recall

plan

1. **Preprocess Data:** Extract code diffs and prepare them for the model.
2. **Augment Data:** Translate Java diffs into C++ using LLMs.
3. **Train Models:**
   - **M_source:** Fine-tuned on real C++ data
   - **M_aug:** Fine-tuned on synthetic (Java → C++) data
4. **Evaluate:** Test on held-out C++ examples, compare results using defined metrics.



Prior ART

Tool	Task solved	Approach/Model	Data Used	Metrics	Results
CodeReviewer (Microsoft)	Automatically classifies code changes as high- or low-quality using transfer learning and real review data	Transformer-based model integrated with GitHub Copilot and Azure DevOps; focuses on PR feedback generation	Microsoft internal codebases, GitHub enterprise repos, developer feedback loops	Precision	Generates review suggestions with 75% precision on internal test sets; 40% of suggestions adopted by developers; improves review coverage by 60%
LLaMA-Reviewer	Performs code review classification by leveraging LLM-augmented, synthetic data for language transfer	LLaMA model fine-tuned via Reinforcement Learning from Human Feedback (RLHF)	Expert-reviewed commits from open-source projects and simulated decision datasets	Accuracy, F1 Score	Matches expert labels in 81% of cases; F1 Score of 0.79; increases reviewer decision confidence by 25%; significantly reduces low-quality approvals
OpenAI Codex	Automatically classifies code changes as high or low quality using few-shot learning	GPT-3-based LLM fine-tuned on code	Public code repositories (mainly GitHub), annotated code review tasks	Accuracy, Precision, Recall, F1-score	Strong zero-shot and few-shot performance; competitive with SOTA

Methodology Overview

How did we try to achieve the objectives?

- **Integrated both real and synthetic data** for model training
- **Leveraged transfer learning** using a language-agnostic pre-trained model (CodeBERT)
- **Established a systematic pipeline** from data preprocessing through evaluation

Pipeline Steps:

1. Data Loading and Preprocessing
2. Exploratory Data Analysis (EDA)
3. Tokenization
4. Training and Fine-Tuning
5. Evaluation

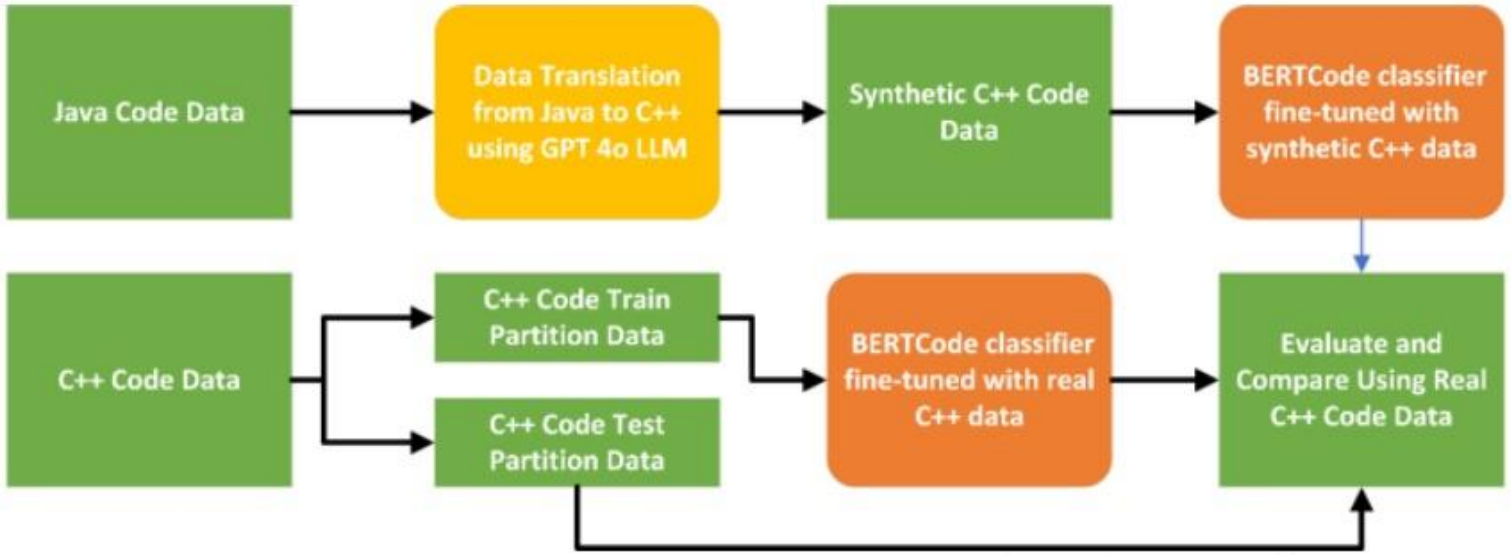


Figure 1: Data generation, training, and evaluation flow

Data Preparation

Data Sources

Real Data:

- C++ and Java code review examples.
- Dataset link – https://huggingface.co/datasets/fasterinnerlooper/codereviewer

Synthetic Data:

- Java code review examples translated to C++ using an LLM GPT4.1.

Labeling:

Each code diff is labeled:

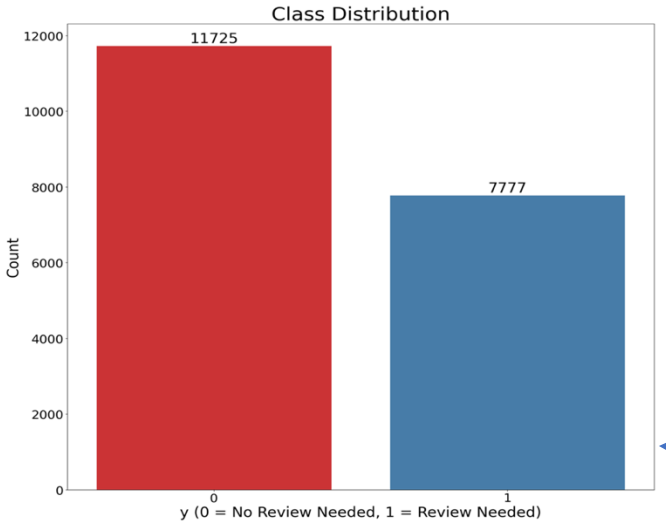
- **0:** High quality, no further review needed.
- **1:** Low quality, requires further review.

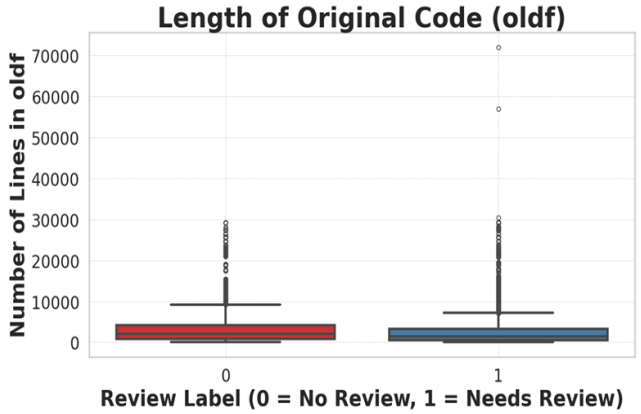
Fields:

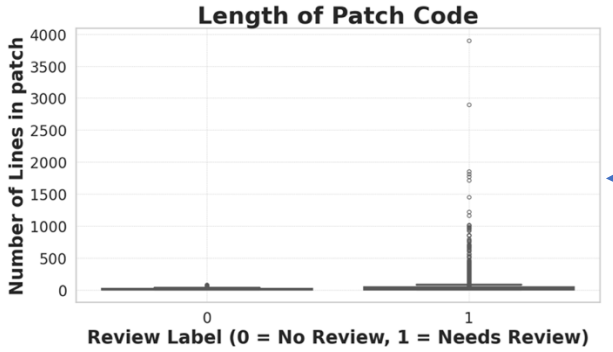
Relevant fields from the dataset:

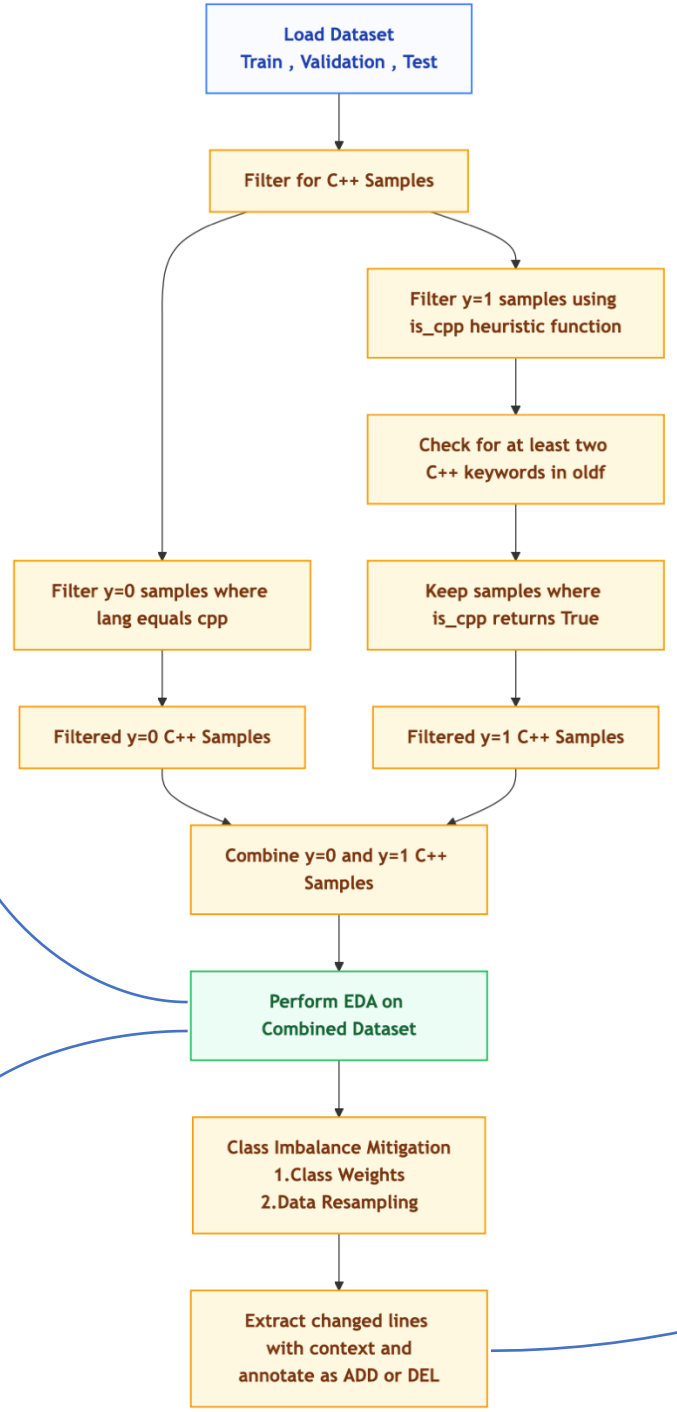
- **Oldf –** The code before the changes.
- **Patch –** The code change.
- **Y –** Label of each example.
- **Lang –** Language of each example (C++,Java , etc.)

Preprocessing – M_Source model









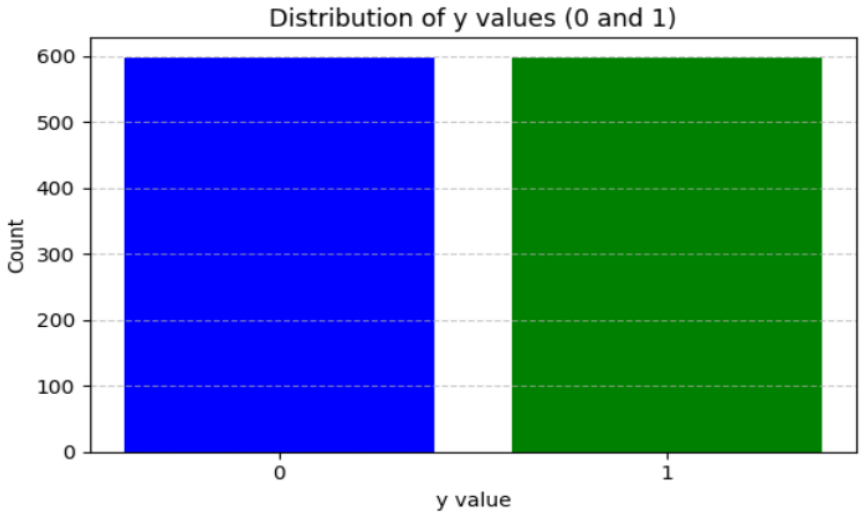
Data Properties

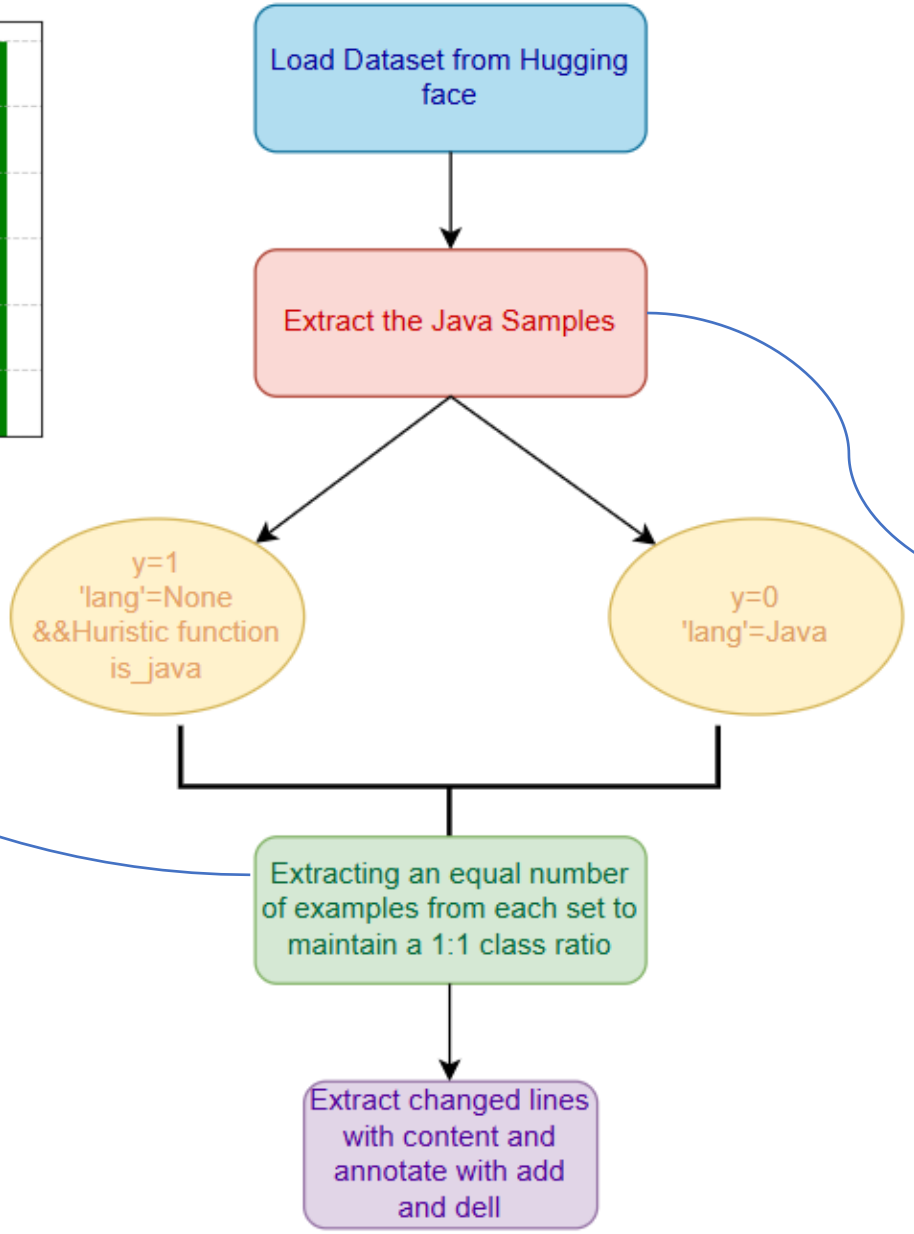
- **Dataset examples:**
  - Train – **23,450**
  - Validation – **3112**
  - Test – **3015**
- **Input length (<= 512 words):**
  - Train set - 96.6% of the dataset.
  - Test and Validation set – 100% of the datasets.

C++ Code Diff Example

```
int process(int x) {
[DEL] Helper helper("process", locals);
[ADD] if (isDynamic()) {
[ADD]     auto out = Dropout::run(x, {
[ADD]         {"prob", 0.5}, {"test", !isTraining()} });
[ADD]     return out;
[ADD] }
[ADD] Helper helper("process", locals);
    return x;
}
```

Preprocessing – M_Aug





Data Properties

- **Dataset examples:**
  - Train – **783**
  - Validation – **173**
  - Test – **239**
- **Input length (<= 512 words):**
  - Train set - 97% of the dataset.
  - Test and Validation set – 100% of the datasets.

Java Code Diff Example

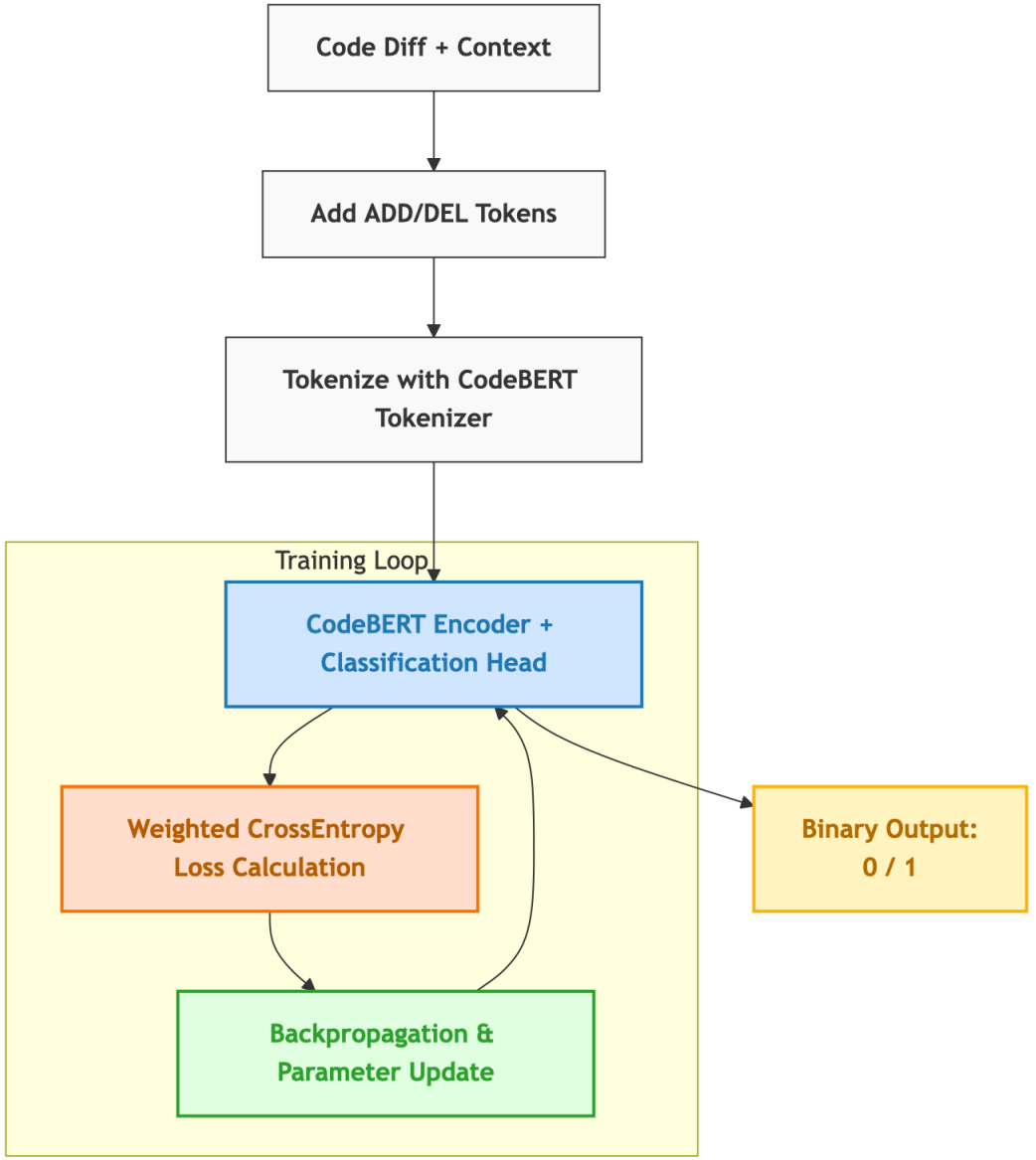
```
Example 3
 import java.util.Map;
 import java.util.SortedSet;

[ADD] import org.osgi.util.promise.Promise;
[ADD]
 import aQute.bnd.osgi.Processor;
 import aQute.bnd.version.Version;
```

Models and Processing Pipelines

M_Source model

- **Model:** CodeBERT
- **Tokenizer:** AutoTokenizer with added special tokens **[ADD]** and **[DEL]**
- **Data split:**
  - **Training**: Stratified resampled + balanced dataset.
  - **Validation**: Held-out original validation set.
- **Training Configuration:**
  - **Trainer:** Custom **WeightedLossTrainer** with class weights to handle imbalance.
  - **Epochs: 8**
    - (Also tested: 6,10,15)
  - **Learning Rate: 2e-5**
    - (Also tested: 3e-5,1e-5,2e-7)
  - **Batch Size: Train=32, Eval=32**
    - (Also tested: 8 , 16 , 64)
  - **Regularization:**
    - Dropout (**rate = 0.3**) in classification head.
  - **Gradient Accumulation: 4 steps**
  - **Warmup Ratio: 0.1**
  - **Weight Decay: 0.01**
  - **Evaluation/checkpointing:** Every 250 steps.
  - **Early stopping:** Triggered after 5 evaluation steps without F1 improvement
  - **Best Model Selection:** Based on F1 score
- **Platform:** Colab



Models and Processing Pipelines

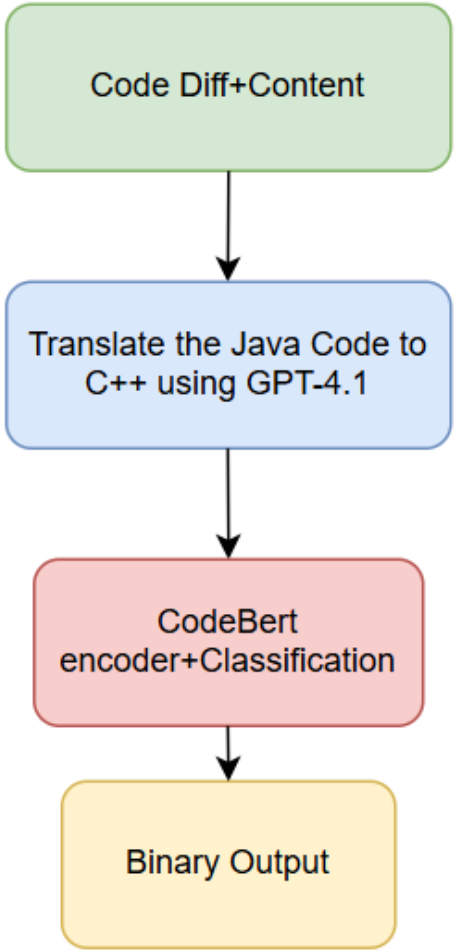
M_Aug model

- **Model:** CodeBERT & GPT-4.1
- Also tested: Starcoder, CodeGemma-2b,CodeGemma-7b
- **Tokenizer:** AutoTokenizer with added special tokens **[ADD]** and **[DEL]**
- **Data split:**
  - **Training**: Stratified resampled + balanced dataset.
  - **Validation**: Held-out original validation set.

  Dataset 1,200 examples — limited due to the extended translation time required.

  Training Configuration:
  - **Trainer: Epochs: 10**
    - (Also tested:15)
  - **Learning Rate: 1e-5**
    - (Also tested:2e-5)
  - **Batch Size: Train=32, Eval=32**
    - (Also tested: 16)

  Gradient Accumulation: 4 steps
  - **Warmup Ratio: 0.1**
  - **Weight Decay: 0.01**
  - **Evaluation/checkpointing:** Every 10 steps, the model's performance is evaluated on the validation set, and every 100 steps, a checkpoint is saved.
  - **Best Model Selection:** Based on F1 score
- **Platform:** Colab PRO



Metrics

Metrics: F1 Score (primary), Accuracy, Precision, Recall

M_Source model

- **During the Training:**
  - Evaluated every 250 steps.
  - Metrics computed on validation set.
  - Used for early stopping (patience: 5 evaluations)
- **During the Evaluation:**
  - Final model evaluated on held-out test set (3015 examples)
- **Additional Notes:**
  - F1 was prioritized due to class imbalance.
  - All metrics were computed with **average='weighted'** to account for label distribution.

M_Aug_model

- **During the Training:**
  - Evaluated every 10 steps.
  - Metrics computed on validation set.
- **During the Evaluation:**
  - Final model evaluated on held-out test set (1200 examples)

Code Organization

GitHub Repository:

- https://github.com/DudiOhayon/CodeReviewer.AI

Code Files Overview:

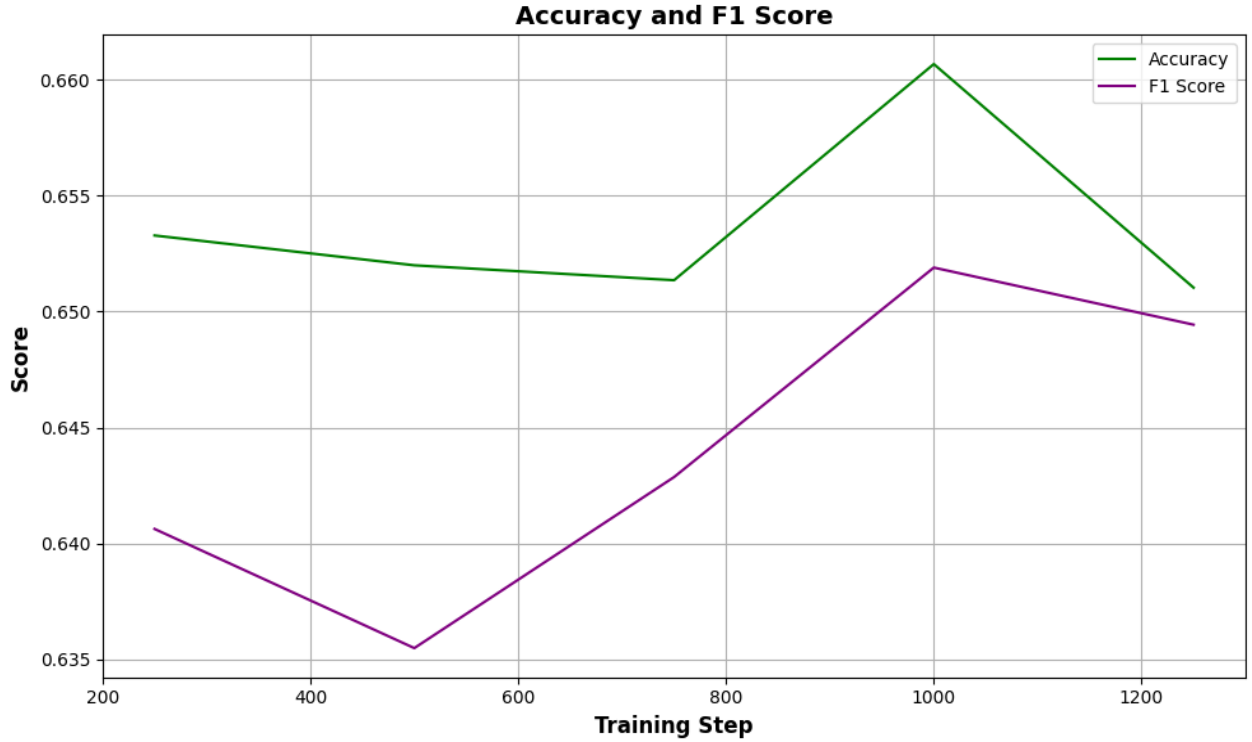
- M_source
- M_aug

Intermediate & Baseline Results

M_Source model

- **Metrics result from Training:**

Step	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
250	0.601600	0.654022	0.653278	0.640624	0.6411960	0.653278
500	0.519200	0.726093	0.651992	0.635486	0.639514	0.651992
750	0.478800	0.685226	0.651350	0.642861	0.641858	0.651350
1000	0.447300	0.762441	0.660668	0.651893	0.651509	0.660668
1250	0.429500	0.759083	0.651028	0.649431	0.648225	0.651028



Intermediate & Baseline Results

M_Aug model

Step	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
10		0.955	0.676	0.676	0.689	0.676
20		0.966	0.682	0.681	0.691	0.682
30		0.995	0.658	0.657	0.679	0.658
40		1.01	0.664	0.663	0.679	0.664
50		0.962	0.670	0.668	0.687	0.67

Main Results

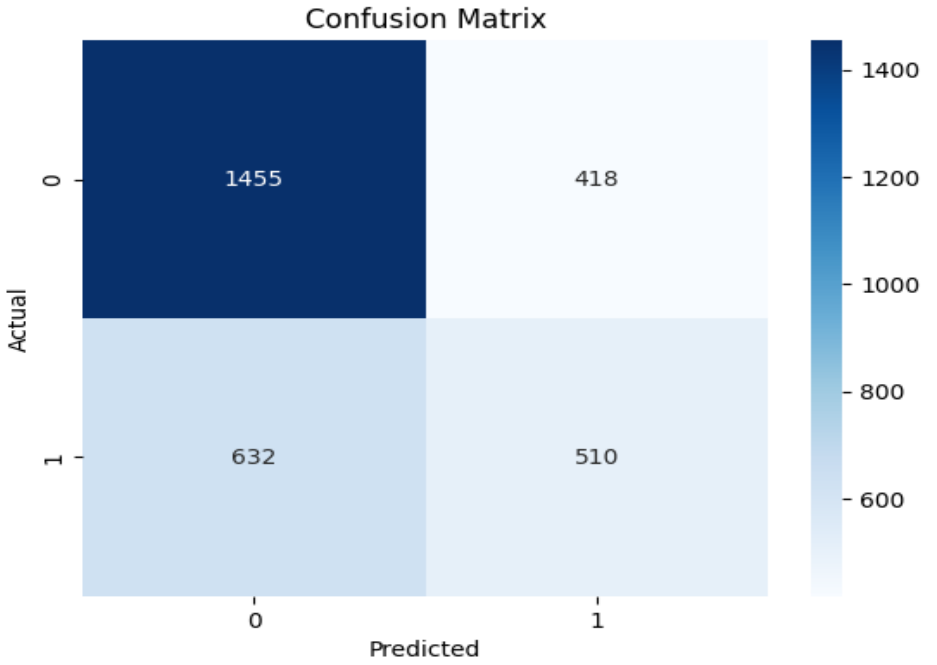
M_Source model

- **Test Set Evaluation Metrics:**

Eval Loss	Accuracy	F1	Precision	Recall
0.773169	0.651741	0.643149	0.641264	0.651741

- **Baseline Result (Without Fine - Tunning)**
  - **Accuracy:** 46.9%
  - **F1 Score:**0.423
  - **Precison:**0.359
  - **Recall:**0.514

- **Effect of the Fine - Tunning:**
  - **Accuracy** improved from 46.9% → **65.1%** (**+38.8%** relative increase**)**
  - **F1** improved from 0.423 → **0.643** (**+52.0%** relative increase)
  - **Precision** improved from 0.359 → **0.641** (**+78.6%** relative increase)
  - **Recall** improved from 0.514→ **0.651** (**+26.6%** relative increase)



Classification Report

	precision	recall	f1 - score	support
0	0.6972	0.7768	0.7348	1873
1	0.5496	0.4466	0.4928	1142
accuracy			0.6517	3015
Macro avg	0.6234	0.6117	0.6138	3015
Weighted avg	0.6413	0.6517	0.6431	3015

- **Conclusion:**
  - **Objective Achieved:** The final model can detect low-quality code changes with **65.1% accuracy** on unseen data.
  - **Balanced Performance:** With F1-score, precision, and recall all in the **0.64–0.65** range, the model demonstrates a well-calibrated decision boundary and consistent predictive behavior.
  - **Significant Improvement:** Compared to the baseline, the model shows substantial relative improvements.

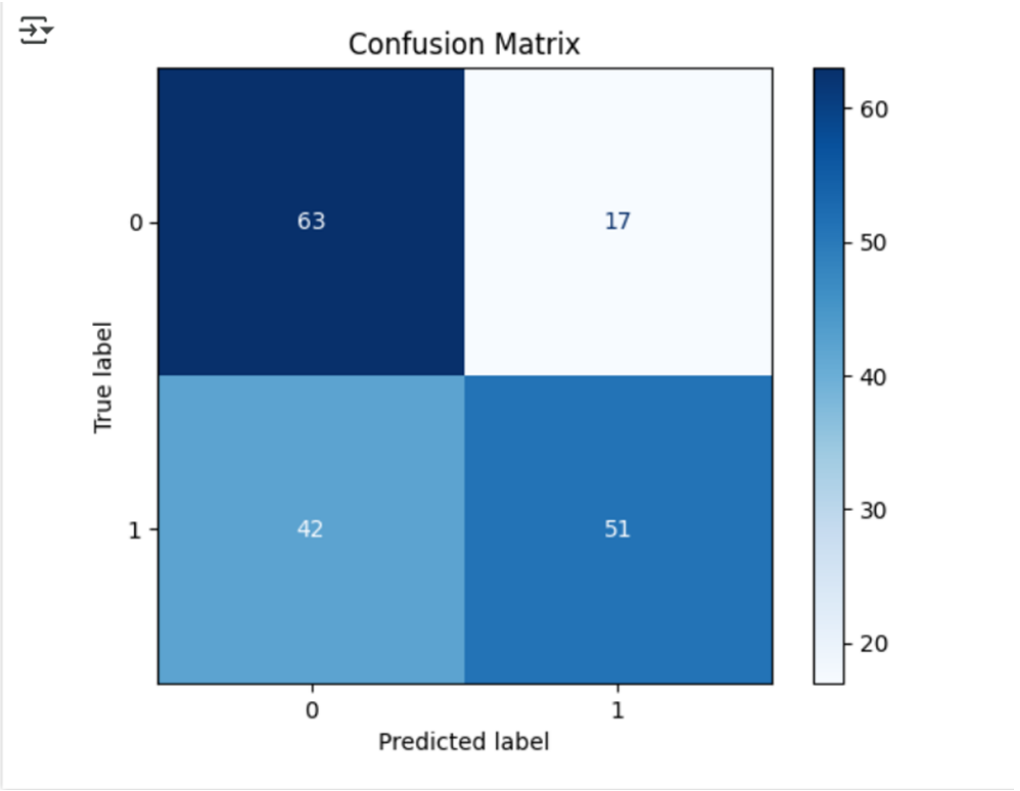
These results underscore the effectiveness of the proposed approach in improving overall classification quality.

Main Results

M_Aug model

- **Test Set Evaluation Metrics:**

Eval Loss	Accuracy	F1	Precision	Recall
1.0467	0.6589	0.655	0.68	0.658



Effects of the Fine-Tuning
The fine-tuning had little impact on the training results, likely due to the limited size of the dataset.

- **Conclusion:**
  - Objective Achieved: The final model can detect low-quality code changes with 65.89 % accuracy on unseen data while being trained on synthetic data.
  - The model achieved moderate performance, with an F1 score of 0.655 and accuracy of approximately 65.89%, suggesting it was able to learn some useful patterns from the translated Java-to-C++ dataset. However, the relatively high evaluation loss (1.0467) and the small dataset size likely limited its overall effectiveness.

Graphical abstract