AI4SE Assignment 1

Talha javed

1. Overview

This project fine-tunes the CodeT5 model for predicting Python `if` statements within code. The workflow consists of six phases: Data Collection, Tokenizer Training, Dataset Generation, Model Pre-training, Model Fine-tuning, and Evaluation. Each phase builds toward a robust, domain-adapted model for conditional statement generation.

2. Data Collection

A large, domain-specific dataset of Python functions was built from open-source repositories on seart-ghs.si.usi.ch, using strict selection criteria:

- At least 100 commits and 10 contributors
- Last commit within two years
- Primary language: Python
- Minimum 100 stars

Repositories were cloned locally to avoid API limits. Python's `AST` module extracted individual function definitions with metadata.

From 373,018 extracted functions, short (<3 lines), long (>100 lines), and duplicate functions were removed. The final dataset contained 250,000 unique functions, stored in `collected_data/python_fu` with fields such as repository name, file path, signature, and tokenized code.

3. Tokenizer Training

A custom Byte-Level Byte Pair Encoding (BPE) tokenizer was trained to efficiently represent Python syntax.

Configuration:

- Vocabulary size: 40,000; Minimum frequency: 3
- Pre-tokenizer: `ByteLevel`
- Special tokens: `<s>, <pad>, </s>, <unk>, <mask>, <if_mask>`

The tokenizer, saved as `tokenizer_files/tokenizer.json`, was used throughout all model stages.

4. Dataset Generation

Two datasets were created—one for Masked Language Modeling (MLM) and another for if-statement prediction.

Parameters: MAX_PRETRAIN = 150K, MAX_FINETUNE = 50K, SEQ_LEN = 512, MLM_PROB = 0.15

4.1 Pre-training Dataset

Functions were tokenized and 15% of tokens were randomly masked. Each record stored masked `input_ids` and `labels`. Output: `pretrain_dataset.csv`.

4.2 Fine-tuning Dataset

In each function, one `if` statement was replaced with `<if_mask>` as the input and the original statement served as the target. Data was split into 80% train, 10% validation, and 10% test, producing: `finetune_train.csv`, `finetune_val.csv`, and `finetune_test.csv`.

5. Model Pre-training

The base model Salesforce/codet5-base was further pre-trained on masked Python data to enhance learn how to predict code.
Configuration:

- Epochs: 1, Batch size: 2 (auto fallback on OOM)
- Precision: FP16; Gradient checkpointing: Enabled
- Output: `./models/codet5-pretrained`

Training adapted CodeT5's representations to Python syntax before fine-tuning.

6. Model Fine-tuning

The pre-trained model was fine-tuned on the if-statement dataset for 3 epochs.
Training Setup:

- Dynamic batch size with OOM fallback
- Evaluation every 500 steps
- Best model saved by minimum validation loss
- FP16 precision and gradient checkpointing enabled

Training resumed automatically from `checkpoint-last`. The best checkpoint was stored at `./models/codet5-finetuned/best_model`.

7. Evaluation

Evaluation used the unseen `finetune_test.csv` and a self-validation set.
Metrics:

- Accuracy: Exact match rate
- Similarity: difflib ratio $\times$ 100
- Perplexity: Model confidence in predicting target text

Dataset	Accuracy	Similarity	Perplexity
Test Set	29.37%	70.83	19.27
Self-Validation	39.62%	73.00	19.12

8. Analysis and Conclusion

Results show that CodeT5 effectively learns patterns in Python `if` statements. Although exact match accuracy remains moderate (30–40%), high similarity scores (>70) indicate semantic alignment. The performance gap between test and validation sets suggests minor overfitting, and a perplexity around 19 demonstrates strong predictive confidence.

This project establishes a full pipeline—from large-scale data collection and tokenizer design to pre-training, fine-tuning, and evaluation—demonstrating the feasibility of adapting CodeT5 for conditional code prediction. The fine-tuned model shows promise for advanced code completion and automated reasoning tasks in programming environments.

9. Acknowledgment

Parts of this project, including code development, debugging assistance, and report writing support, were completed with the help of AI tools such as ChatGPT and Google Gemini.