GenAI for Software Development: Assignment 2

Comfort Ohajunwa
ccohajunwa@wm.edu

1 Introduction

In this assignment, I fine-tune a CodeT5 model from Hugging Face to suggest if-conditions in Python functions. CodeT5 is a Transformer model pre-trained on a large corpus of code in various programming languages for code understanding and generation tasks. Specifically, I fine-tune a smaller variant, `codet5-small`, which has approximately 60 million parameters. The goal is for the fine-tuned model to predict the if-condition in a given Python function, where the if-condition is replaced by a special token. For this work, I prepared a training dataset consisting of around 50,000 clean Python functions. I also modified the training, validation, and test datasets through masking and tokenization. The model was then fine-tuned using the training and validation datasets. Finally, I evaluated the fine-tuned model on the test set using exact match, BLEU-4, and CodeBLEU. The source code for this project can be found at `https://github.com/cohajunwa/CodeT5-for-If-Statements`.

2 Implementation

2.1 Dataset Preparation

GHS Repository Selection: I use the GitHub Search tool (`https://seart-ghs.si.usi.ch/`) to compile a set of GitHub repositories with the following filters: language = "Python", minimum number of stars = 100, minimum number of commits = 100. These criteria yield a total of $31,295$ repositories, from which I randomly selected 150 to prepare the dataset.

Extracting Functions: I wrote a script called `create_dataset.py` to pull the repositories and extract Python functions. The Pydriller package is used to retrieve Python source code from the selected repositories, and the built-in ast package is used to extract the functions, which are compiled into a Pandas DataFrame.

Pre-processing: The `create_dataset.py` script also preprocesses the DataFrame by removing comments, filtering out duplicate functions, excluding those with non-ASCII characters, and discarding outliers based on function length. Finally, it formats the DataFrame to have three columns: `clean_method` (cleaned Python functions), `target_block` (target if-conditions), and `tokens_in_method` (number of tokens in the function). From the 150 repositories I selected, I extracted $840,505$ Python functions. After preprocessing, the final training set contained $50,435$ functions.

2.2 Fine-Tuning Process

I use a script called `fine_tune.py` to implement the fine-tuning process, including modifying the datasets, loading the model, tokenizing the data, and training.

Modifying Datasets: For the fine-tuning process, I modified the training (created using the process outlined in Section 2.1), validation, and test sets by first masking if-conditions and then flattening the functions. During masking, the target if-statement is replaced with a special token: `<IF-STMT>`. During flattening, newlines are eliminated, and tabs are replaced with a special `<TAB>` token to preserve indentation.

Model Loading and Tokenization: In addition, the pre-trained `codet5-small` model and tokenizer are loaded from Hugging Face. The `fine_tune.py` script also adds the new `<IF-STMT>` and `<TAB>` tokens to the tokenizer. The datasets are then tokenized for fine-tuning.

Model Training: The model is set to train for seven epochs using early stopping on the loss function of the validation set. After training, `fine_tune.py` stores the tokenized dataset and the best-performing model locally. These files are later used for evaluating the model.

2.3 Evaluation

Metrics: I evaluated the fine-tuned model on the test dataset using three metrics: exact match, BLEU-4, and CodeBLEU.

Exact match simply checks whether the predicted if-condition is the same as the actual if-condition.

BLEU is a natural language processing metric that measures the quality of machine-generated text against reference text. BLEU-4, in particular, measures 4-gram precision, meaning it evaluates the similarity between predicted and reference texts by considering the overlap of n-grams up to four tokens in length. I used the sacrebleu Python library to compute the BLEU-4 score of the predicted Python function (with the generated if-condition).

CodeBLEU is a variant of BLEU that is more tailored towards code generation by accounting for structural and semantic features of programming languages. For computing the CodeBLEU scores, I adapted some source code from Microsoft's CodeXGlue project (`https://github.com/microsoft/CodeXGLUE`). Specifically, I took their source code for evaluating CodeBLEU (originally designed for command-line use) and modified it to be callable within my script `evaluate.py`.

I wrote a script `evaluate.py`, which loads the fine-tuned model and tokenizer as well as the tokenized dataset. For each masked Python function in the test set, it generates a function with its predicted if-condition, then evaluates the prediction using exact match, BLEU-4, and CodeBLEU. In the end, it produces a CSV file called `testset-results.csv` containing the scores for all the Python functions in the test set.

Results: The model predicted an exact match for none of the inputs in the test dataset. This result is because the exact match metric is sensitive to minor syntactical differences, such as whitespace. The BLEU-4 and CodeBLEU scores provided more helpful insights. The average BLEU-4 score is 40.60, and the average CodeBLEU score is 95.33. The CodeBLEU score is much higher likely because it accounts for programming syntax and semantics. Consider the following case:

- Expected: `if output == input :`
- Predicted: `if input == output:`

Both if-statements are essentially the same. However, the model's prediction had a BLEU-4 score of 54.48, but a much higher CodeBLEU score of 97.17.