GenAI for Software Development: Assignment 2

Gigi Kuffa jkuffa@wm.edu

Kimberly Sejas kmsejas@wm.edu

Eden Fitsum efitsum@wm.edu

1 Introduction

In this assignment, we aim to finetune the CodeT5 Transformer model from Hugging Face to automatically recommend suitable if statements in Python functions.

CodeT5 is a pre-trained encoder-decoder model optimized for code understanding and generation tasks, leveraging its identifier-aware training to excel in syntactic and semantic code comprehension. Specifically, we employed the lightweight CodeT5-small variant, which balances computational efficiency with robust performance, containing approximately 60 million parameters.

Our work involved preparing the given dataset of Python functions by masking the if conditions and flattening the code to ensure compatibility with the model's input requirements. The dataset was then tokenized using the pre-trained tokenizer associated with CodeT5 before fine-tuning the model on this processed data. Finally, we evaluated the model's performance using multiple metrics to comprehensively assess its ability to predict missing if conditions.

The source code for our work can be found at https://github.com/jdkuffa/cs420-assignment2.

2 Implementation

2.1 Dataset Preparation

Loading Pre-Trained Model: We began by loading the pre-trained CodeT5-small model using the Hugging Face library. This smaller version of CodeT5, with 60 million parameters, balances computational efficiency and performance.

Masking & Flattening Code: To prepare the dataset for fine-tuning and simplify tokenization, Python functions were first modified by masking the if conditions with a placeholder token <MASK>. This masking ensures that the model focuses on predicting the missing condition based on the surrounding context. Additionally, all functions were flattened into a single-line format to align with the input requirements of CodeT5.

Code Tokenization: The pre-trained tokenizer provided with CodeT5 was used to tokenize the modified dataset into a sequence of tokens that the model could process. The placeholder token used to mask was preserved during this step to maintain compatibility with downstream tasks.

Model Training: The training was conducted using early stopping to prevent overfitting. Early stopping monitored validation loss after each epoch and halted training when performance on the validation set plateaued or deteriorated over consecutive epochs. This approach ensured optimal generalization while avoiding excessive memorization of training data.

Model Evaluation: The performance of the fine-tuned model was evaluated using multiple metrics to ensure comprehensive assessment, including exact match, BLEU-4, and CodeBLEU metrics. The model demonstrated strong performance across all evaluation metrics, but particularly excelled when measured by CodeBLEU **(44.64)**, reflecting its ability to capture code semantics and structure beyond surface-level token similarity. The model also achieved a BLEU-4 score of **42.3** and an exact match rate of **22%**, indicating both syntactic fluency and partial alignment with ground truth outputs.