Rachel Zheng
rzheng02@wm.edu

Gen AI for Software Development: Assignment 2

1 Introduction

In this assignment, we built a model for the task of predicting if statements in Python. Specifically, the model takes as input a function containing a special token, <MASK>, masking a single if condition and will attempt to predict it. We used the CodeT5 transformer model, a pre-trained encoder-decoder Transformer model designed for code understanding and generation and fine-tuned it for the purpose of predicting if statements. The source code for our work can be found https://github.com/rachelzheng03/Predicting-If-Statements.

2 Implementation

2.1 Dataset Preparation

We started with three datasets of pre-processed data provided by Prof. Mastropaolo. The training dataset consists of 50,000 training samples and the validation and test sets each contain 5,000 samples each. However, the target if statements needed to be masked, and the methods needed to be flattened.

Flattening: To flatten the methods, we first tokenized the methods using the Pygments Python package. This way we can identify and easily remove the newlines ("\n"). Since Python relies on indentation to determine code blocks, we also identified the tabs in this step and replaced them with the special token <TAB>. We decided that every four consecutive spaces at the beginning of a line would be considered a tab. However, our data also contains a version that disregards tabs, so that we can test if adding the <TAB> token affects the model's performance.

Masking: We replaced the first occurrence of the target if-statement in the flattened method with the special token <MASK> using Python's string method, replace.

Lastly, in order to make our data compatible for the training of the model, we converted our data into a DatasetDict object.

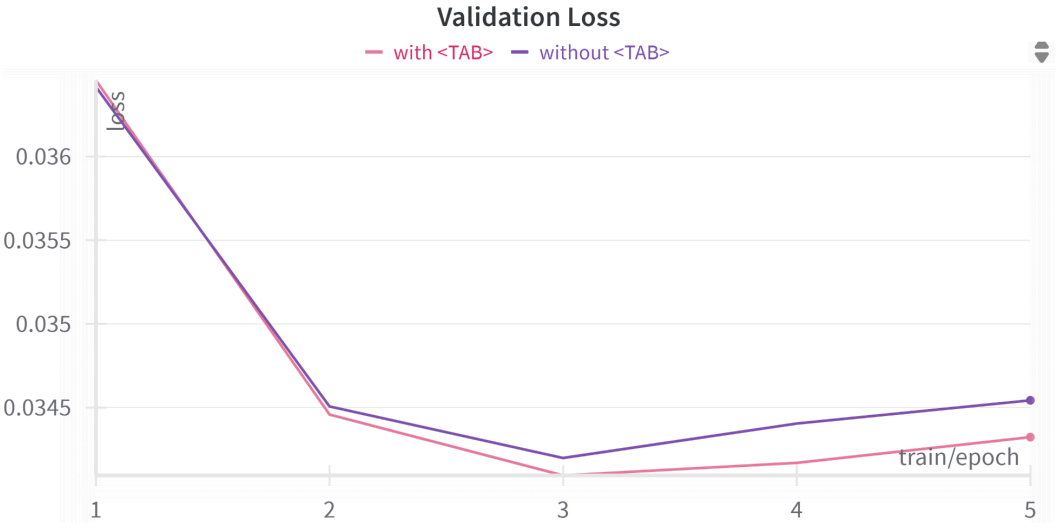
2.2 Model Fine-Tuning

Fine-Tuning: As mentioned earlier, we fine-tuned our model on the CodeT5 transformer model, specifically codet5-small. Before beginning the fine-tuning, we tokenized our data with a preloaded tokenizer called the RobertaTokenizer. We set the model to train for six epochs and used early stopping so that the model would stop training if the loss of the validation set increased over two consecutive epochs. This helped to avoid over-fitting and under-fitting. We

Rachel Zheng
rzheng02@wm.edu

used wandb to visualize different aspects of the training process such as validation loss over epochs and GPU power usage.

Training Results: We trained two models, one whose inputs have the <TAB> token and the other whose inputs do not. For both the models, the training stopped early at five epochs. The figure below shows early stopping in action as we can see that the validation loss increased twice in a row. In addition, the figure also demonstrates that the model with the <TAB> tokens performed better on the validation data.



2.3 Evaluating Our Model

We evaluated our model on the test dataset. First, we used our fine-tuned model to predict the if condition. Then, we calculated three different evaluation metrics for each prediction: CodeBLEU, BLEU-4, and exact match. We used the CODEXGLUE code provided in class and SacreBLEU to get the CodeBLEU and BLEU-4 scores, respectively.
For CodeBLEU, we set the weight of each of the four components that make up the score (ngram match, weighted ngram match, syntax_match, and dataflow_match) to 0.25.

Results and Model Comparison: We computed the average CodeBLEU and BLEU-4 scores and the number of exact matches across our two models. From the table below, we can see that the model with the <TAB> token performed better in regards to the BLEU-4 score and number of exact matches, while the model without the <TAB> token performed better with respect to the CodeBLEU score.

Rachel Zheng
rzheng02@wm.edu

	CodeBLEU	BLEU-4	# of exact matches
With <TAB>	88.85	96.44	1466
No <TAB>	93.25	94.12	1432