Jaden Kingsley
04/10/25
GenAI for Software Development
Assignment 2

**Fine-Tuning CodeT5 for Predicting if Statements**

In this project, I fine-tuned the smaller CodeT5 model, called “codet5-small”, to predict masked “if” conditions in Python functions. The goal was to train the model to recognize patterns in code and generate an “if” statement based on the context.

The dataset I used was already provided pre-processed, with 50,000 training samples split into training, validation, and testing sets. Each sample included a “cleaned_method” (a flattened Python function) and a “target_block” (the “if” condition that needed to be predicted). I created a new version of each function, called “masked_method”, by replacing the “if” condition with a “<mask>” token. Once masked, I tokenized both the input functions and target conditions using a pre-trained tokenizer from Hugging Face. I ended up testing with only 50 elements in each set (training, validation, and testing), due to very lengthy runtimes.

For training, I used Hugging Face's Trainer class, and trained for up to 5 epochs. I also added early stopping based on validation loss to avoid overfitting - the training stopped if the model didn’t improve for two epochs in a row. The best-performing checkpoint was saved for evaluation. To evaluate, I ran the model on the test set. For each example, the model received the masked function and predicted what it thought the missing “if” condition should be. I used three metrics to assess performance:

**Exact Match Accuracy:** This checks if the predicted condition matched the actual one exactly. The model achieved an exact match accuracy of **36%**. This is a fairly low score, which I believe can be attributed to some issues regarding empty predictions, along with the smaller corpus used (as mentioned earlier), but does not necessarily indicate that the model is poorly adapted to the task. The individual exact match scores (indicated as true or false) can be found in the output file (testset-results.csv), or in ExampleResults.csv .

**BLEU-4, using SacreBLEU**: This metric checks how many 1-to-4-token sequences match between the predicted and actual conditions. It doesn't account for code syntax or structure, but

it's a good way to measure surface-level similarity. The model's average BLEU-4 score was **58.36**, and some example individual scores for each prediction can be found in ExampleResults.csv, or by running the code and opening testset-results.csv . The average BELU-4 score is relatively high, which indicates that the model is decently fine-tuned to the task, but could be improved upon.

**CodeBLEU**: This is a more advanced metric that evaluates predictions based on syntax, data flow, and tree structure. The CodeBLEU evaluation metric is based on a scale of 0-1, with 1 indicating perfect predictions, and 0 indicating a poorly-adapted model. Unfortunately, I ran into issues computing CodeBLEU scores for the individual predictions. However, I was able to get an overall average CodeBLEU score across the entire model, which was approximately **0.1543**. This score indicates that the model is somewhat capable, but there is much room for improvement.

Overall, using a mix of evaluation metrics, I was able to see that the model can benefit from additional fine-tuning, and I would be curious to see how the scores change if I was able to use the entire dataset. In my opinion, the CodeBLEU score was the most useful indicator of the model's capability, because it is a more advanced metric. The BLEU-4 score was somewhat misleading, because a 58.36% score is much higher than the relatively low-range scores that CodeBLEU and Exact Match Accuracy gave. Exact Match Accuracy can be useful, but taking into account the specifics is necessary when building an AI model - an answer can be merely somewhat correct or incorrect, but Exact Match Accuracy does not account for that. I'm fairly satisfied with this project, and believe I have learned a lot about evaluative metrics and how to implement them.