**Dataset Preparation/Preprocessing:**

The original dataset (stored in ft_train.csv) consists of 50,000 Python methods as well as a specific if-statement within each method that the model is being trained to guess. We use masking as a pre-processing technique while creating the training set, where we change the condition in the specified if-statement to a token <mask>. A parallel column is also kept where we just have the original statement as well. The column with the if condition replaced with the mask serves as the input column, while the value of what should be in place of the mask serves as the label.

Masking the input that needs to be solved for works because transformers are trained to understand dependencies in code over long contexts. Because we replaced the conduction with <mask>, this encourages the model to infer what logically fits in the position of the mask using semantic clues from the rest of the given function. Even without the original condition, the context around the rest of the method will often give enough information for a model to reconstruct it.

Additionally, in the preprocessing step we additionally look to flatten the function bodies as well as remove indentation and newline characters. This simple processing of the tokens allows for more efficient training.

**Fine Tuning/Training Process**

The pre-trained base model used in the project is CodeT5-small, a smaller variant of the T5 model which is specifically pre-trained on programming tasks. In its pre-trained state, CodeT5 has already learned to understand general patterns of code, such as syntax, logic flows, and variable usage. However, the point of pre-training is to look for patterns among a vast dataset and not exactly worry too much about any output. This is why we need fine-tuning.

Fine-tuning allows us to train a pre-trained model on a smaller dataset to perform specialized tasks related to the pre-training dataset. In our case, the specific task is teaching a model to predict values of conditional statements using the smaller dataset of methods and their conditionals, while the pre-training dataset first allows a model to understand how patterns in code work over large corpuses of code.

I used Google Colab to fine-tune the model and I trained it for 5 epochs as well with early stopping. This would allow enough time to train, but would also allow for the model to stop training if it was starting to overfit the data based on the results of checking the model against the validation set. Checkpoints would be saved at the end of each epoch, and the model with the lowest validation lost would be restored at the end for final evaluation. The model was created so that it would take in a method (missing a conditional statement replaced with a mask token) and return its predicted autocomplete for what should go instead of the mask token.

**Results/Evaluation**

To start, the model would look at all the examples in the testing set, and first find their predicted output for the missing conditional statements. We would then compare the ground truth versus predictions of the test set and then calculate evaluation metrics which we could use to judge the model.

**Exact Match Rate:** The Exact Match Rate of the model was 58.98%, which means the model performed at almost 60% accuracy on examples it had never seen before. I would say this is really good, and it shows the model did do a good job of learning to infer what should go in conditionals.

**BLEU-4 and CodeBLEU scores:** The model scored 63.56 on Bleu-4 and 69.61 on the CodeBleu scores. These are extremely high scores and show that the predictions are very close to the ground truth on a regular basis.

**F1 Score -** The model scored .78 on the F1 metric, which indicates that it was getting most of the important tokens, which would make the predictions useful in practice.

Altogether, based on the metrics it seems that the model in fact did perform well in predicting the conditional statements, and that we did accomplish our goals of creating a good model to do this task.