GenAI for Software Development Assignment 2

Ella Bryant

egbryant01@wm.edu

# 1 Introduction

The CodeT5 Fine Tuned Model sought to generate the correct if-statements that correctly complete the code. The fine-tuned model modifies the small CodeT5 model by training the model specifically for generating if statements. Specifically, this was completed by masking the target if-statements in a data set, tokenizing the data, training the model using an early stopping evaluation metric, and performing the code generation on the testing data. Finally, evaluations of the model’s performance were completed using CodeBLEU, BLEU-4, precision, recall, and F1 score. The source code can be found at [ViolettGee/CodeT5\_Fine\_Tuning\_For\_If\_Statement\_Prediction\_In\_Python](https://github.com/ViolettGee/CodeT5_Fine_Tuning_For_If_Statement_Prediction_In_Python).

# 2 Dataset Preparation

**Dataset.** We started by using the dataset pre-prepared in the “Archive” folder, containing “ft\_train.csv” with 50000 data points, “ft\_test.csv” with 5000 data points, and “ft\_valid.csv” with 5000 data points.

**Preprocessing.** First, the preprocessing of the dataset was completed. To do so, each of the “csv” files were initialized into Panda’s data frames for better data analysis. Each of the rows of each of these files was iterated through with the following data processes. Using the Pygments Python Lexer, the methods and target code sequences were both flattened, and white space was made uniform. The target section in the method was then replaced with the mask token “<MASK>” using the string replace function. After the processing, of each row was completed that rows data was saved to the respective new data frame. The final results were then exported to the “Processed\_Data” folder to the files: “training.csv”, “testing.csv”, and “validating.csv” respectively.

**Tokenization.** Next each of the “csv” files in “Processed\_Data” containing the preprocessed data were initialized into Pandas data frames for better data analysis. Each of the rows of each of these files was iterated through with the following tokenization process. Using the Python RobertaTokenizer section of the Transformers library, the tokenizer corresponding to the small Code-T5 model was loaded. This tokenizer was then used to encode and decode each of the methods and targets. The process readied the model to receive data of the pattern it would expect, and it would help to optimize expected and predicted later in model evaluation.

# 3 Model Training

**General Implementation.** The original model was taken from the Huggingface CodeT5 small model found at: [Salesforce/codet5-small · Hugging Face](https://huggingface.co/Salesforce/codet5-small). The model was then fine-tuned using the processed and tokenized dataset described above. More specifically, the model was further trained using that data. To complete this, the training arguments were first initialized containing the file save location, evaluation strategy (“epoch”), and learning rate (0.0005). Using the RobertaTokenizer section of the Transformers library, the tokenizer was loaded, and the model was loaded using the T5ForConditionGeneration section of the Transformers library. The data collator was then initialized from the Transformers library, specifically DataCollatorForSeq2Seq since it is specialized for labeled data. The early stopping function is then initialized form the Transformers module, so the training will avoid overfitting and underfitting the data. Then, the tokenized training and validation datasets are loaded in using the Datasets Library from the “Tokenized\_Data” folder, labeled as “training.csv” and “validating.csv” respectively. Each of the data points from the dataset were then embedded using the tokenizer described above. The tokenized method column was used as the input ids for encoder training and the tokenized target column was used as labels for decoder training within the model. The training and validation data sets were then set in the format of “torch” using the Pytorch library. Then, the trainer was initialized with all of the values described above before running the fine-tuning. When completed, the model was saved into the “Model” folder, which was too large to include in the github repository and can be found at <https://drive.google.com/drive/folders/1HJ04K5MgQ4qAzWtYtUGTCOzTaBLHDUp7?usp=sharing>.

# 4 Model Evaluation

**Model Testing Implementation.** The goal was then to run the test data points through the model to evaluate the effectiveness of the fine-tuned model. First, the tokenizer was loaded using the RobertaTokenizer section of the Transformers library, and the fine-tuned model was loaded on using the “checkpoint-18750” version since it is the model with all the training completed. Then, the tokenized testing dataset “testing.csv” was loaded in from the “Tokenized\_Data” folder into a Pandas data frame. Using the testing data frame, each of the rows is analyzed using the following process to compute outputs. The tokenized method is embedded using tokenizer to get the models input ready. The embedded method is then passed to the model and the output is decoded, so it is in a string format. This output is the predicted if-statement. The model then computes if the predicted sequence is an exact match to the expected sequence outputting “Yes” and “No” respectively. Using the Codebleu module, the CodeBLEU score is calculated comparing the expected and predicted sequences of the current row. The BLEU-4 score is calculated comparing the expected and predicted sequences of the current row using the Evaluate and Sacrebleu modules. The data of that row is then added to the output Pandas data frame into “input\_method”, “exact\_match”, “expected\_if\_condition”, “predicted\_if\_condition”, “CodeBLEU\_score” and “BLEU-4\_score” respectively. Once each of the data points is computed through in the testing dataset, the output Pandas data frame is exported to the “output.csv” file.

**Evaluation Metrics.** Using the “output.csv” file, the average CodeBLEU score, average BLEU-4 score, precision, recall and F1 score were computed. For computation, the “output.csv” file was imported into a Pandas data frame. Using the mean function, the columns containing the already computed CodeBLEU score and BLEU-4 score. The resulting values were 0.45042 and 46.43017 respectively. Next, the counts for true positives, false positives, true negatives, and false negatives were computed. In the scope of our model, the goal was to have the model correctly generate the masked if-statement. Therefore, we can define a positive value as a generated sequence containing an if-statement, and a negative value as a generated sequence not containing an if-statement. The outputs Pandas data frame was filtered into two smaller data frames containing the sequences with positive values and negative values respectively. Using the re library to search the output sequences for “if”, those containing “if” statements were filtered into the positives data frame, and those without were filtered into the negatives data frame. Then, the true and false values were defined as the correct or incorrectly generated sequences respectively. Since it is too time consuming to go through each value and define whether it is correct or not, the true and false values were computed based on how similar and dissimilar the predicted sequences were to the expected sequences based on the CodeBLEU scores. Those positive and negative data frames were separated into the false and negative values. Those with a CodeBLEU score higher than 0.45 were classified as true, and those with a CodeBLEU score less than 0.45 were classified as false. Afterwards, each of the remaining filtered data frames only have the data points that are classified as true positive, false positive, true negative and false negative respectively. Therefore, the values can be computed by pulling the number of rows from the data frames. These values were then used to compute precision, recall and F1 score resulting in values of 0.4154, 1.0, and 0.58697 respectively.