GenAI for Software Development: Assignment 2
Rachel Huang (rhuang03@wm.edu), Jackson Taylor (jttaylor01@wm.edu)

1) The Fine-Tuning Process

We are fine-tuning the small version of CodeT5, a pre-trained encoder-decoder Transformer model, to be able to predict missing if conditions in Python functions. The model takes as input a dataset containing functions that mask their if conditions with a special token. The model then tries to predict the masked token. We load the pre-trained CodeT5 model from Hugging Face, and then tokenize our dataset with the RobertaTokenizer. We then fine-tune the model, avoiding overfitting and underfitting by using early stopping on the loss function of the validation set. Finally, we evaluate the model's performance with scikit-learn's accuracy, precision, recall, and f1 scores, as well as BLEU and CodeBLEU scores. The source code for our work can be found at https://github.com/jtaylor05/genai_project_2/.

2) Dataset Preparation

Dataset Selection: We use the provided pre-processed dataset which was pre-split into the three groups: 5000 test methods, 50,000 train methods, and 5,000 validation methods. Each entry in the data set included three parts: the whole method under the header "cleaned_method," the if-statement to be masked under the header "if-statement" and the number of tokens in the method under the header "tokens_in_method."

Cleaning: To clean each dataset, we used the file *mask_methods.py*. The first method used to clean was masking the if statements. To do so, each method in the "cleaned_method" column is split around the string of tokens given as the if-statement. The two halves of the method are then recombined around the <MASK> token. The next step was to remove all special characters \r and \n to flatten the dataset. The last step added the <TAB> token. Since there was no special character \t used in the cleaned_method data, the heuristic that 4 spaces is equivalent to one \t was utilised.

Code Tokenization: We tokenized our dataset with the RobertaTokenizer and added the tokens used during cleaning: <TAB>, <MASK>.

3) Evaluation Results

We implement customized evaluation metrics with the HuggingFace Trainer, specifically recording our fine-tuned model's accuracy, precision, recall, f1, BLEU, and CodeBLEU scores.* Due to aforementioned trouble completing the training process on the full datasets, our metrics are the result of training on 5000 methods, testing on 100 methods, and validating on 100 methods. We observe an eval_loss score of 0.030879, an eval_accuracy score of 0.992226, an eval_recall score of 0.992226, an eval_f1 score of 0.991876, and an eval_bleu score of 0.0. Though our Bleu score was 0.0 due to the decreased size of our datasets, we did apply the sacrebleu 'floor' smoothing method with a smooth_value of 0.1. The counts and the totals array that contributed to the Bleu score calculation were [834, 20, 1, 0] and [1038, 35, 3, 0], respectively. These metrics were computed after each epoch, and our early stopping on the loss function implementation for preventing overfitting and underfitting meant that for this specific training run, these metrics were calculated after the third epoch completed. The complete metrics results can be found below.**

* Our model's accuracy, precision, recall, and f1 scores were calculated with Python's Scikit-learn library, and our BLEU score was calculated with HuggingFace's sacrebleu metrics.

**{'eval_loss': 0.030879925936460495, 'eval_accuracy_score': 0.9922269570707071, 'eval_recall_score': 0.9922269570707071, 'eval_f1_score': 0.9918769927596857, 'eval_bleu_score': {'score': 0.0, 'counts': [834, 20, 1, 0], 'totals': [1038, 35, 3, 0], 'precisions': [80.34682080924856, 57.142857142857146, 33.333333333333336, 0.0], 'bp': 0.9961538509042884, 'sys_len': 1038, 'ref_len': 1042}, 'eval_runtime': 16.8033, 'eval_samples_per_second': 5.892, 'eval_steps_per_second': 2.976, 'epoch': 3.0}