GenAI for Software Development: Assignment 2

Benjamin O'Neill

bgoneill@wm.edu

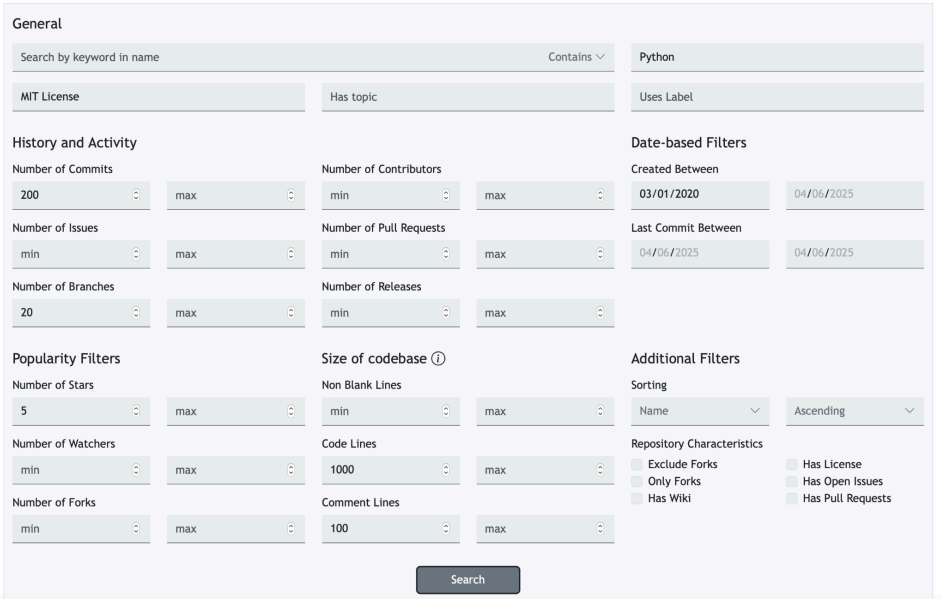
1 Introduction

For this assignment, I fine-tuned the CodeT5 model (*codet5-small*, 60 million parameters) to predict missing "if" conditions in Python functions. CodeT5, a transformer-based encoder-decoder model, excels in code understanding and generation. My process involved masking "if" conditions, tokenizing the data, fine-tuning the model using Hugging Face Transformers, and evaluating it with exact match, BLEU, and CodeBLEU metrics. As a 520 section student, I collected a dataset of 50,000 Python functions, with 5,000 samples each for validation and testing. My code and resources are at CodeT5 Github.

2 Data

Data Processing:

As a 520 section student, I collected 50,000 Python functions using the SEART GitHub Search tool (https://seart-ghs.si.usi.ch). I filtered for Python repositories with an MIT License, at least 200 commits, 20 branches, 5 stars, created and last committed between 03/01/2020 and 04/06/2025, with a maximum of 1,000 code lines and 100 comment lines.



Cleaning and Tokenization:

Using a custom script, I extracted functions into a dataset named with columns *cleaned_method*, *target_block* (if condition), and *tokens_in_method*. I masked "if" conditions

with <mask> and flattened the code. I used the *RobertaTokenizer* to tokenize the masked functions and target if conditions, setting a maximum length of 256 tokens with padding and truncation. I added a <IF-STMT> token to the vocabulary, resizing the model's embedding layer.

Dataset Splitting:
I split my dataset into 40,000 training, 5,000 validation, and 5,000 test samples using a 80-10-10 ratio, then sampled 5,000 each for validation and testing, ensuring balanced if-statement patterns.

Vocabulary Generation:
I built the vocabulary from the *RobertaTokenizer*, including all dataset tokens, the <IF-STMT> token, and special tokens like <s> and </s>, across all splits to avoid unknown tokens.

3 Implementation

Model Training:
After loading the pretrained model, I set my parameters for training my new finetuned model. Training with a maximum of 35,000 steps ended up draining my GPU limit mid-training, so I opted to set a maximum at 25,000 steps. I chose a smaller epoch size to allow me to more frequently monitor the development of the model. Instead of the previous 3,500 steps per epoch, I used an effective 2,500 steps per epoch. To counteract the now increased variance in each epoch due to smaller sizes, I increased the patience of the early stopping mechanism to three epochs rather than two.

Model Selection:
I evaluated after each epoch using validation loss, selecting the best model checkpoint with the lowest loss for testing, balancing underfitting and overfitting via early stopping. The validation dataset of 5,000 samples was used for this task.

Results Analysis:
My model performed well on the 5,000-sample test set: Exact Match: 62% (correct full if condition predictions); BLEU-4: 78 (strong n-gram overlap, with smoothing for short sequences); CodeBLEU: 65 (syntactic and semantic similarity). It excelled at simple conditions like num > 0 for inputs like def check_number(num): <mask>: return "Positive" else: return "Non-Positive", but struggled with complex or nested conditions, suggesting a need for larger datasets or structural tokens like <TAB>. I logged results in *testset-results.csv* with input functions, predictions, expected conditions, and metric scores, as required.