GenAI for Software Development: Assignment 2

Whitt Sanders
wjsanders@wm.edu

Kayla Anderson
kganderson01@wm.edu

Arya Shryock
ijrayshryock@wm.edu

1 Introduction

If-statement Predictor aims to automatically predict if statements in Python code. In this assignment, We take the pre-trained Salesforce/codet5-small model and fine-tune it using our collected data. The results of the training are automatically recorded to Weights & Biases recording prominently the loss and learning rate as the epochs progress. Finally, we evaluate the performance of the model using exact match, f1 score, codeBleu, and Bleu-4 scores. The source code for our work can be found here.

2 Implementation

2.1 Dataset Preparation

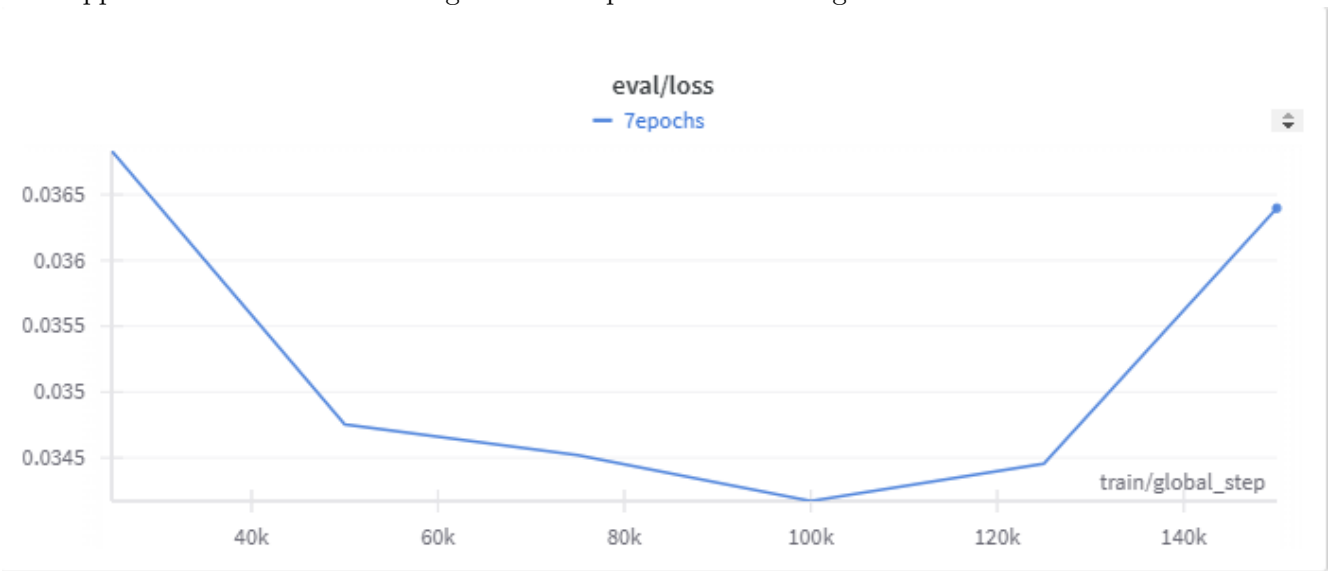
Dataset Preparation Process: To prepare the data set for fine-tuning, we created a pre-processing file. Our file takes a csv as input, strips all newlines from the `cleaned_method` column, and replaces the text representation of the tab spacing in the methods with a `<tab>` token to preserve python indentation. Since the values in the `target_block` column were already tokenized, we could not simply run a replace function on the `cleaned_method` column. To handle this, we created a list of all the values in the `target_block` column and stripped them of spaces. Then, we used a regex function to search each method for the target agnostic of spaces, and replaced the first instance of the target block with a `<IF-STMT>` token. After these processing steps, we saved the output in a new csv.

Code Tokenization: We utilize RoBERTa to tokenize the Python methods to achieve parity with the Salesforce/codet5-small model.

Weights & Biases Usage: Since our training logs to W&B by default, user login is required. Details for setup are elaborated upon in the `README.md` file.

2.2 Model Training Methodology

Model Training & Evaluation: We fine-tune the codet5-small model to fit our dataset using the training arguments found in the main function of `ifstmt-predictor.py`. We allow for 7 epochs of 25000 steps each, but our model had the lowest loss at the 4th epoch. To prevent the model from over-fitting, we stopped the model from training after two epochs of increasing loss.



Our Best-performing Model: Our best-performing model can be downloaded here. This link expires on May 5th, 2025, as required by William & Mary, but the same model can be generated with the data

provided in the repository.

Model Testing: Using our best-performing model, we generate predictions for the entire test set. These predictions along with their analysis can be found in `testset-results.csv`. An example of this behavior is provided below. The average scores of our model for the entire test set were; an exact match 31.34% of the time, `codeBleu` = 88.53, `Bleu-4` = 41.62, and `F1` = 0.6350. Overall the model performed fairly well, often returning an exact match or a similar prediction. The `codeBleu` score was particulary high, because in this case the entire method was used as input with either the predicted or ground truth; this was in order for the data flow match to be evaluted. This lead to a high score as many tokens were shared simply from the input method.

2.3 Model Evaluation Explanation

Evaluation Process: Evaluation of the model was done on `Google Colab`. The code we used for evaluation can be viewed in the `outputIfStat.ipynb` file. The zipped model was uploaded along with the test set for evalutation. We looked at 4 different metrics; exact match (1 for true, 0 for false), `CodeBleu`, `Bleu-4`, and `F1` score. `CodeBleu` was calculated using the code from Microsoft's CodeXGLUE repo, `CodeBleu` is an combination of 4 different metrics, which we weighted equally (25% per metric). These are ngram match, weighted ngram match, syntax match, and data flow match. `Bleu-4` was calculated using sacrebleu, which itself returns the Bleu-4 score on solely the if statement comparison. The F1 score was calculated using sklearn's f1 module; evaluating at a token level (not character level), and using the "macro" average (which means the metric is calculated for each label, and returns the unweighted mean).

Example: Given the input method of:

```
def listdir(self, d):<tab>try:<tab><tab>return [
<tab><tab><tab>p<tab><tab><tab>for p in os.listdir(d)
<tab><tab><tab><IF-STMT><tab><tab>]<tab>except OSError:<tab><tab>return []
```

the model predicted

```
if os . path . isdir ( os . path . join ( d , p ) )
```

when the ground truth was

```
if os . path . basename ( p ) != "CVS"
and os . path . isdir ( os . path . join ( d , p ) )
```

Newlines were added to the above example for readibility.

The scores for this particular case were exact match = 0 (not a match), `codeBleu` = 74.00, `Bleu-4` = 45.94, and `F1` = 0.2657.