Assignment 2
Training a Transformer Model for Predicting *if* statements

By Chuntian Chi, Matthew Chen, Danny Otten, and Kate Balint

Introduction

The *if* statement is one of the most ubiquitous and useful programming statements in computer science. It is the first statement with a condition taught in many computer science courses and serves as a fundamental concept for all programmers. According to the paper An analysis of programming language statement frequency in C, C++, and Java source code, object-oriented languages like C, C++, and Java, *if* statements are the third most common programming statement. For a language like Python, which puts less emphasis on objects and more frequently used in data science, *if* statements are likely even more common due to their omnipresence in tasks like removing outliers, interpolating missing entries, and performing sorts on the data.

Therefore, programmers would be greatly benefited by a Transformer model that is capable of generating useful *if* statements. Our first step is to obtain suitable code to fine-tune our model with. We chose a Hugging Face Dataset containing 503,502 Python functions for this assignment. The corpus was collected and preprocessed by Husain et al. in their paper Evaluating the State of Semantic Code Search using libraries.io. The search criteria required that the GitHub repositories compiled be publicly available, licensed to allow distribution, open-source, referenced by at least one other project, and not forked from another repository. The data is then sorted in order of popularity, putting projects that more users have starred and forked first. This ensures that only industry-standard, high-quality, and non-duplicate projects with a practical application are incorporated as a part of the collection.

Processing took place with function granularity and any functions that were likely to be copy-pasted, outliers compared to the average function, or generally low quality were excluded. This included functions that were shorter than three lines, designed for testing purposes as indicated by a test in the function's name, constructors like __init__, or class extensions like __str__. Duplicate functions are also a cause for concern, and were removed according to the method in The Adverse Effects of Code Duplication in Machine Learning Models of Code. Each function's tokens and literals were separated into sets. If two functions had a Jaccard similarity of 0.8 or greater for their token sets and a Jaccard similarity of 0.7 or greater for their literal sets, they were marked as duplicates and excluded from the data.

Dataset preparation

The dataset construction process involves parsing and processing Python functions to prepare two datasets: one for pre-training with random token masking and another for fine-tuning with masked if-statement conditions.

Parsing and Tokenization

We first parse Python code from each function in the dataset and identify functions containing at least one if statement (this resulted in 282973 Python functions in total). We used the AST (Abstract Syntax Tree) module to parse Python code and extract functions containing if statements, which captures the structure of Python code better than plain string matching. 200k functions were selected as our full dataset. Then, each extracted function is tokenized using the LLaMA tokenizer. The tokenization includes setting a maximum length and truncating the code where necessary to prepare for the input for model training.

Pre-training Data Preparation

130k functions were used for pretraining, which is the largest training dataset we can make run on our available GPUs by testing different sizes of training datasets. In the pre-training stage, functions undergo random token masking, where 15% of tokens are randomly replaced with a <MASK> token in each function. This helps the model learn general patterns in Python code syntax and structure. In this step, we tokenize functions first to ensure that each token is recognized consistently by the model's vocabulary. Then, we mask individual tokens. This preserves the vocabulary structure and allows the model to learn consistent patterns.

Fine-tuning Data Preparation

70k functions were used for fine-tuning. For fine-tuning, we mask only the conditions in if statements, replacing them with the <MASK> token. When testing, we realized that by "if condition" the instructions intended the "if" and ":" to be included, but while creating the datasets we interpreted this as purely the True/False portion.

The AST module is also used for extracting if-conditions. This is to train the model specifically on identifying and completing if conditions. This dataset is used to teach the model to predict conditions in masked if statements. In this step, we mask the if conditions first before tokenization, so that the model can learn to predict the <MASK> placeholder based on surrounding tokens. Nested if-statements were evaluated individually. Masking before tokenization ensures that <MASK> is treated as a distinct token that the model will recognize as something to be filled in, and it learns the context of where if statements appear and the structure around them. Among the 50k

fine-tuning data, 80% were used as training data for fine-tuning; 10% were selected as evaluation dataset to decide when should we stop the model training, and the last 10% were used as a test dataset for generating predicted if conditions using our resulting model after fine-tuning.

Model Training

Model Setup and Initialization

We used a T5-based Transformer initialized with weights from the Salesforce/codet5-small checkpoint. We use the LLaMA tokenizer for tokenization and a data collator to ensure proper padding during training.

Pre-training

For pre-training, the model learns from the pre-training dataset with randomly masked tokens to improve its general understanding of Python syntax. Hyperparameters used:

- Batch size: 1
- Epochs: 3
- Logging: Every 10 steps

Fine-tuning

Once pre-training is complete, we fine-tune the model on the fine-tuning dataset. In this step, the model learns to predict masked if conditions, developing the capability to recommend appropriate if statements.

Results

While the model did not generate any exact matches, it did use terms that were in the sample if conditions, such as ‘self,’ ‘path,’ ‘ast,’ ‘body,’ and ‘id.’ This demonstrates that the model learned from and attempted to implement the given vocabulary.

Unfortunately the format of the provided testset is not what our model was designed for, so the model failed to make use of the provided tokens and output 26 underscores in response to every prompt.

GitHub Link

https://github.com/dsotten/AI4SEproject2/tree/main