Training a Transformer model for Predicting Condition Statements

Md Mehedi Hasan Sun, Md Robiul Islam

Department of Computer Science
The College of William & Mary
Williamsburg, Virginia, USA

Abstract. In recent years, there has been a growing need for tools that can analyze and generate code automatically, helping to improve software development speed and quality. This project focuses on training a Transformer model to predict conditional statements in Python methods, especially "if" statements. We use a two-step training approach: first, we pre-train the model on Code Search Net Dataset where 15% of the tokens are randomly masked. This helps the model learn the relationships and patterns in Python code. Next, we fine-tune the model on the dataset that focuses on predicting complete "if" statements. We found that this method is effective in the model's ability to suggest suitable "if" conditions. Our source code is available at github and our pre-train model and custom dataset also available in Google Drive at Drive

Keywords: code analysis, python methods, mask, pre-training, fine-tuining, if condition.

1 Indtroduction

Pretrained models based on Transformer architectures[1] have resulted in cutting-edge performance across a wide range of NLP tasks They can generally be classified into three categories: encoder-only models, which include BERT[2], RoBERTa[3], and ELECTRA[4] ; decoder-only models such as GPT[5]; and encoder-decoder model such as T5[6]. In the era of rapid technological advancement, the field of software development has increasingly embraced automation to enhance efficiency, reduce human error, and streamline workflows. As programming languages evolve, developers are often faced with complex decision-making constructs, particularly conditional statements like "if" conditions, which play a critical role in determining the flow of execution within code. The ability to accurately predict and generate these constructs not only aids in writing more robust software but also assists developers in understanding and modifying existing codebases. Recent advancements in natural language processing (NLP), particularly through the application of Transformer architectures, have shown promise in tasks that involve understanding and generating human languages. These models have revolutionized text processing by leveraging large-scale datasets and self-attention mechanisms to capture contextual relationships within data.

This capability has raised the question of whether similar techniques can be effectively applied to programming languages, which, despite their structured nature, share many syntactic and semantic features with natural languages. In this project, a pre-trained model CodeT5 is used for pre-training and finetuining for specific task which is predict if condition in our case. In summary, we make the following contributions:

- We used a pre-training methodology that involves masking 15% of input tokens in Python methods. This approach allows the model to effectively capture the patterns and structures of programming syntax, thereby enhancing its understanding of conditional logic.
- The project employs a fine-tuning strategy that specifically targets the prediction of entire "if" conditions. By using a dataset designed for this purpose, the model is trained to not only recognize but also generate contextually relevant and syntactically accurate conditional statements.

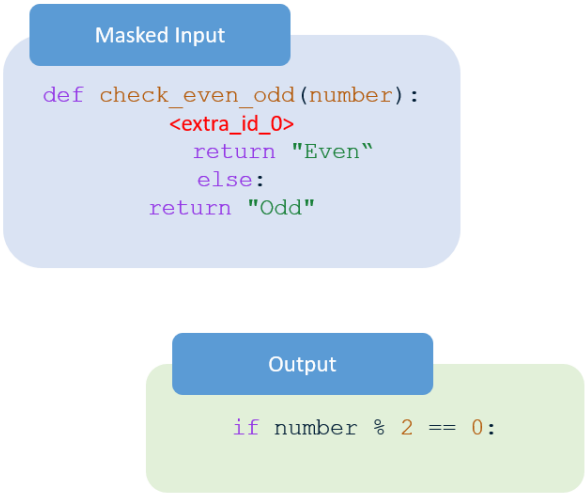

Fig. 1. Masked if prediction

2 Data Collection & Pre-processing

For every research data is very important. In this assignment, we need almost 200k of data where we'll use 150k for training the model and 50k for testing the

model. To collect the data we use SEART website. We use only 2 main parameters to collect the github repositories, first of all the number of contributors should be at least 100 and the code lines should be more than 1000 linesand obviously we selected python as a language. We use this because we want to ensure that we can collect larger projects and our procedure will be easy.

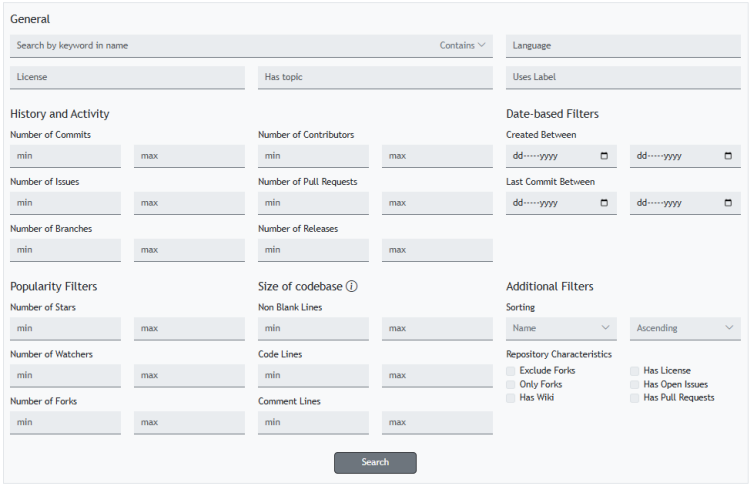

Fig. 2. SEART website

2.1 Data Collection

While collecting the data from the SEART website. We found out that there is a public dataset available for doing our project. For simplicity of the model, we've used that dataset which is called CodeSearchNet[7]. We've used 80% for training the model and 10% for testing the model and finaly 10% for validation of the model.

2.2 Data Pre-processing

Our data preprocessing steps follows:

– **Test method filtering**: We checked if there is any test method in our dataset. We removed the test method. In our custom dataset, there are around 10% test methods. but in Code Search Net Dataset [7] we found no test methods.

– **Comment Removal**: We remove the comments both single-line comments as well as multiple-line comments.
– **Docstring Removal**: We found many functions used docstring in two way. For i.e. triple quoted string and raw string (r"'..."'). We removed both types of docstring from our dataset.
– **White Space Refactor**: After the preprocessing mentioned above, we observed that many functions contained uneven and excessive whitespace, such as multiple newlines. We refactored these newlines to a single occurrence or allowed at most two occurrences.

We performed more preprocessing but differently in both the pretraining and fine-tuning phases. we discussed that in our *Methodology* section.

3 Methodology

3.1 Model Pre-training

For pre-training the model, we selected the CodeT5-base model from the Hugging Face repository of pre-trained models [8]. This model has been pre-trained on a large dataset of various programming languages. We utilized the *Masked Language Modeling* (MLM) objective to further pre-train our model.

Data Preparation: Since there are over 400K Python methods in Code Search Net Dataset [7]. We selected 40% of data (Nearly **200K**) to pre-train our model. We have splited in a manner that train data, validation data, and test data consist of respectively 80%, 10% & 10% datapoints. For pre-training the model, we used a default value of 15% random masking of our tokens using **extra_id_0** mask token.

Pre-training: In table 1, we've discussed the parameters we've used to pre-train our model. We use **cross-entropy loss** as the default loss function of our training. Since we have large dataset, we evaluated our model after every 500 steps not after epochs. We also used **mixed precision training** to improve performance and reduce memory usage. For monitoring our model performance and training, we used **Weights & Biases** (WandB). We implemented an **Early Stopping Mechanism** to stop training when the model shows no improvement after consecutive iterations, setting the patience parameter to 3.

Evaluation: Finally, to assess the model's performance in context learning, we **evaluated** it both before and after pre-training. We hypothesized that the model would not perform well in predicting masked tokens prior to pre-training. Our findings showed that the **BERTScore** for predicting masked tokens increased by **15%**. Specifically, the model exhibited a **BERTScore** of **78%** before pre-training, which rose to **93%** after the pre-training was completed. We used the same dataset and masking tokens to ensure the accuracy of our comparisons.

Table 1. Parameter for pre-training the model

Parameter name	**Value**
num_train_epochs	3
evaluation_strategy	"steps"
eval_steps	100
save_steps	500
learning_rate	1e-5
logging_steps	1e-10
per_device_train_batch_size	4
per_device_eval_batch_size	4
num_train_epochs	1
save_total_limit	4
load_best_model_at_end	True
fp16	True
report_to	"wandb"

3.2 Model Fine-tuning

We used our pre-trained CodeT5-Base model to fine-tune it for the infilling task. Our aim is to predict the conditions that we masked within the function.

Data Preparation: From the Code Search Net Dataset [7] we selected another 10% of data (Nearly **45K**) to pre-train our model. We have splited in a manner that train data, validation data, and test data consist of respectively 80%, 10% & 10% datapoints. For pre-training the model, we randomly masked of one single condition from each method string using **extra_id_0** mask token.

Pre-training: In table 2, we've discussed the parameters we've used to finetune our model. We use the default loss function of our training. For monitoring our model performance and training, we used **Weights & Biases** (WandB). We use 3 epochs and starting learning rate of *1e-4* to train our model.

Evaluation: Finally, to assess the model's performance in context learning, we **evaluated** it using our testing dataset. We calculated the prediction score using softmax activation layers output.

4 Conclusion

This study has explored the application of the Code-T5 model in automated code generation, specifically targeting the prediction of "if" statements in Python

Table 2. Parameter for fine-tuining the model

Parameter name	**Value**
num_train_epochs	3
learning_rate	1e-5
save_total_limit	3
evaluation_strategy	"epoch"
save_strategy	"epoch"
per_device_train_batch_size	4
per_device_eval_batch_size	4
weight_decay	0.01

methods. By using a two-stage training process that includes pre-training with masked tokens and focused fine-tuning on conditional statements, we have demonstrated the model's ability to learn and generate syntactically accurate and contextually relevant conditions and can be used in more downstream infilling tasks

5 Acknowledgements

We would like to thanks Joseph Hause (Computer Science Systems Administrator), Kate Balint (Techies) for their support to increase the disk size in the system.

References

1. A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.
2. J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.
3. Y. Liu, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, vol. 364, 2019.
4. K. Clark, "Electra: Pre-training text encoders as discriminators rather than generators," *arXiv preprint arXiv:2003.10555*, 2020.
5. A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.
6. C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.
7. H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "CodeSearchNet challenge: Evaluating the state of semantic code search," *arXiv preprint arXiv:1909.09436*, 2019.
8. Y. Wang, W. Wang, S. Joty, and S. C. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," *arXiv preprint arXiv:2109.00859*, 2021.