Assignment 2 : CodeT5: Training a Transformer model for Predicting if statements

Afia Farjana

[1]Williamsburg, Virgina

`afarjana@wm.edu`

1. Abstract

This assignment explores the fine-tuning of the CodeT5 model to enhance its ability to predict if conditions within code snippets, building upon a base of specialized pre-training focused on identifier naming. The pre-training phase involved a custom Masked Language Modeling (MLM) strategy, where identifiers were specifically masked to train the model in understanding and predicting their context within Python code. Fine-tuning was conducted over 10 epochs using a meticulously prepared dataset, with conditions masked to test the model's predictive accuracy. Key optimizations included using fp10 precision and a gradient accumulation strategy to maximize computational efficiency. The outcomes were evaluated through checkpoints, that demonstrated modest gains in accuracy, essential for tasks such as code completion and automated suggestions in development environments.

2. Introduction

In the dynamic field of software development, artificial intelligence (AI) plays a crucial role in streamlining code generation and comprehension tasks. The use of deep learning models, such as CodeT5, introduces advanced techniques for analyzing and manipulating code syntax and semantics. This research focuses on optimizing CodeT5 to enhance its ability to predict if conditions in Python code. It employs a specialized pre-training strategy using Masked Language Modeling (MLM) to improve the model's understanding of identifier names, followed by fine-tuning on a targeted dataset designed to refine its logical condition handling. This dual-phase approach not only bolsters the model's syntactic accuracy but also its applicative performance in coding environments, paving the way for more sophisticated automated code assistance technologies

3. Methodology

3.1. Pre-training Setup

My methodology involved a specialized pre-training setup utilizing the CodeT5 model, which was primed specifically to enhance its proficiency in understanding code through a Masked Language Modeling (MLM) approach, with a particular focus on identifier names. This pre-training was pivotal for adapting the model to accurately understand and predict the usage of identifiers such as variable and function names within code blocks. During this phase, identifier names were masked at a 100% rate, replacing each with a <mask> token. This approach was not merely to challenge the model but to compel it to decipher the crucial contextual clues that govern identifier usage in programming languages.

The rigorous masking regime was coupled with an enhancement of the tokenizer to include <tab> as a special token, ensuring that structural and syntactic aspects like indentation were preserved and accurately represented in the tokenized data. This targeted approach was designed to deepen the model's contextual understanding of code, significantly enhancing its capabilities in advanced code completion task. This strategic focus on identifiers, rather than broad variable names.

3.2. Model Training and Fine-Tuning

After the initial pre-training, the model underwent a fine-tuning phase on the CodeX-GLUE Python dataset, which was meticulously prepared to align with the task of predicting if conditions within code. This dataset was formatted with masked if conditions represented by <extra_id_0> tokens, placing the model in scenarios very similar to real-world code manipulation tasks. Fine-tuning CodeT5 on this dataset allowed the model to apply its pre-trained knowledge in a targeted manner, focusing specifically on enhancing its ability to predict conditional logic accurately—a crucial skill in software development and debugging.

3.3. Benefits of Specialized Pre-training

The focused pre-training on identifier naming significantly enhanced the model's contextual understanding, allowing it to excel not just in recognizing but in predicting the placement and naming of variables and functions within unseen code snippets.

4. Data Preparation

4.1. Pre-training Preparation

CodeT5 model, focusing on the predictive accuracy of identifier names within code blocks. This dataset was derived from a carefully selected collection of Python repositories, chosen based on stringent criteria including the presence of at least 500 stars, 500 commits, 500 issues, and 50 watchers, ensuring a wide representation of coding styles and patterns while excluding duplicates. From over 3,000 potential repositories, 100 were randomly selected for in-depth analysis and data extraction. Utilizing the Method_extractor.py script, essential data such as method signatures, documentation, and project metadata were extracted using abstract syntax tree (AST) parsing. The processed data was then compiled into a CSV file before being converted into a JSON format, named Dataset.json, containing more than 32,000 method entries specifically structured for machine learning applications.

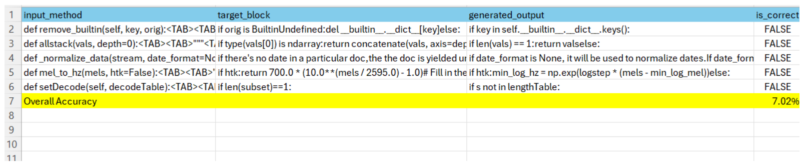
Tokenization and dataset preparation were meticulously executed to enhance the model's training efficiency and effectiveness. Non-essential elements like comments and whitespace were removed, and the RegexpTokenizer was used to decompose the code into tokens, preparing it for the masked language modeling (MLM) phase. The dataset was strategically divided into training, validation, and test subsets to ensure exposure to varied code complexities, which is critical for developing a model that generalizes well across different coding environments. .

4.2. FineTuning-Preparation

Fine-tuning for If Condition Prediction Dataset Preparation: For the fine-tuning phase, the CodexGlue Python dataset was utilized along with a specifically prepared CSV file containing two columns: input_method and target_block. I utilized a sample of 4,516 data points from the CodeXGLUE dataset. The dataset was strategically divided using an 80-10-10 train-validation-test split to optimize the model's learning and generalization capabilities. The input_method featured the original method code annotated with <tab> for indentation, preserved blank spaces, and <extra_id_0> to signify masked tokens where if conditions were masked and target_block represents only original if condition.

5. Results & Discussion

The CodeT5 model underwent a focused evaluation after being fine-tuned over 10 epochs, with each checkpoint specifically tested for its ability to predict masked if conditions within code snippets. To enhance training efficiency, the model was fine-tuned using fp10 precision optimization, a training batch size of 1, and a gradient accumulation across 8 batches. This setup was critical in managing computational resources while ensuring adequate learning over the epochs.



input_method	target_block	generated_output	is_correct
def remove_builtin(self, key, orig):<TAB><TAB	if orig is BuiltinUndefined:del __builtin__.__dict__[key]else:	if key in self.__builtin__.__dict__.keys():	FALSE
def allstack(vals, depth=0):<TAB><TAB>"""<T	if type(vals[0]) is ndarray:return concatenate(vals, axis=dep	if len(vals) == 1:return valselse:	FALSE
def _normalize_data(stream, date_format=Nc	if there's no date in a particular doc,the the doc is yielded ur	if date_format is None, it will be used to normalize dates.If date_forn	FALSE
def mel_to_hz(mels, htk=False):<TAB><TAB>'	if htk:return 700.0 * (10.0**(mels / 2595.0) - 1.0)# Fill in the	if htk:min_log_hz = np.exp(logstep * (mels - min_log_mel))else:	FALSE
def setDecode(self, decodeTable):<TAB><TA	if len(subset)==1:	if s not in lengthTable:	FALSE
Overall Accuracy			7.02%

Figura 1. Best accuracy among three checkpoints

Performance assessment was conducted through three saved checkpoints, evaluated against a split test dataset, particularly emphasizing the model's prediction accuracy at each stage. The results, as recorded in separate CSV files for these checkpoints, showcased accuracies of 4.02%, 3.72%, and 7.02% respectively. This progression illustrates a modest yet crucial development in the model's capability to handle and predict complex conditional statements accurately, which is vital for real-world coding applications where such predictive proficiency directly enhances code completion and intelligent suggestion systems.

6. Conclusion

The experiment confirmed that fine-tuning CodeT5 with a targeted approach on identifier prediction and if condition generation offers a promising avenue to improve AI-driven code understanding and manipulation tools. Despite modest accuracies in the initial trials, the method showed potential in enhancing the model's contextual awareness and prediction capabilities. Further research could explore deeper neural network architectures or more extensive datasets to amplify the model's performance. Ultimately, this research lays groundwork for leveraging advanced machine learning techniques to refine the interaction between coders and development environments, potentially leading to more intelligent, efficient, and user-friendly coding tools.

Note: I used Chatgpt for writing this script and some research paper for getting idea about coding implementation