Analysing Code-BERT and its efficacy for the code-comment relationship prediction task

Mr. Akshay Dongare
Department of Computer Engineering,
Marathwada Mitra Mandal's College
Of Engineering, Savitribai Phule Pune
University, Pune, India
akshayd02@gmail.com

Dr. Kalpana Thakre
Department of Computer Engineering,
Marathwada Mitra Mandal's College
Of Engineering, Savitribai Phule Pune
University, Pune, India
kalpanathakre@mmcoe.edu.in

Abstract: In AI-powered development, exploring machine learning techniques has become pivotal to enhancing the understanding and interaction between developers and software. This study delves into the efficacy of CodeBERT, a pre-trained model tailored for programming languages, specifically examining its performance in predicting the relationship between code snippets and associated comments. Python notebooks serve as a unique learning opportunity due to their utilization of Markdown annotations to elucidate the programmer's intent, providing valuable insights into the rules governing classes and functions. The primary objectives of this research include the development of an artificial intelligence model capable of predicting the intricate relationship between code annotations and the corresponding code itself. We employ the Kendall correlation metric to assess the model's effectiveness, offering a quantitative measure of the alignment between predicted and ground truth relationships. Drawing inspiration from recent advancements in NLP, particularly CodeBERT, our analysis builds upon the dual-model architecture that combines pre-trained programming language and natural language components. This study aims to unravel how CodeBERT, with its Transformer-based neural architecture, interprets the nuanced interplay between natural language descriptions and programming code.

Keywords—NLP, CodeBERT, Code-Comment Relationship Prediction, Kendall correlation, Transformer, Python Notebooks.

I. INTRODUCTION

Computers and other devices are highly beneficial when handling data or tabular information. However, individuals frequently express themselves through spoken or written sentences instead of using tables. Human speech and writing often contain excessive information, making it challenging for computers to interpret. Natural language processing (NLP) aims to enable computers to comprehend non-standard text and extract valuable information. NLP is a subfield of artificial intelligence that focuses on the interaction between computers and humans. This study aims to comprehend the relationship between code and description in Python scripts. To achieve this, we will rearrange the order of Markdown units in a document to match the order of code units, thereby unveiling the natural language's understanding of the involved code.

The research objectives encompass:

1. Developing an artificial intelligence model capable of predicting the relationship between annotations and code.
2. Assessing the prediction model's performance using the Kendall correlation.

II. LITERATURE REVIEW

Feng et al. colleagues [2] proposed CodeBERT, a pre-trained model for programming and languages.
CodeBERT is a dual-model, pre-trained programming language (PL) and a natural language called CodeBERT (NL). CodeBERT provides language searching, document coding, etc. It learns global representations to handle low-level NL-PL applications such as To determine the appropriate choices drawn by the generator, CodeBERT is trained using a combinatorial objective function that combines pre-training functions on known token transformations. CodeBERT is built using Transformer-based neural architecture. This allows the user to use both single-mode data and bi-mode NL-PL pair data; the latter helps to better identify the generator, and the former is used as the signal for standard training. Two NL-PL implementations using CodeBERT were evaluated by tuning the parameters. As a result, CodeBERT leads projects that require data generation and natural language search. Additionally, data for NL-PL detection was generated and analyzed in a zero-shot scenario, where the parameters of the pre-trained model were further tuned to investigate the type of information obtained in CodeBERT. According to the results, CodeBERT performs better than pre-trained models in NL-PL detection.
Phan et al. al [3] introduced CoTexT: Multidisciplinary learning using CodeText Transformer.
CoTexT is a preliminary encoder-decoder model developed on Transformer. Examines the content representation of programmatic and natural language (NL) (PL). CoTexT is pre-trained in the general organization of programming languages, using self-monitoring to gain a good understanding of the language and programming code. CoTexT supports downstream NL-PL functions such as code

generation, error checking, debugging, and code collection and annotation. CoTexT was trained using a variety of PL corpuscles, including "bimodal" and "unimodal" documents. In this case, single-mode data contains only code snippets, but bi-mode data contains text and associated code snippets. Small and medium-sized features from the CodeXGLUE database were used in CoTexT's first multi-learning evaluation that also included Code Refinement.
Detailed testing was also performed to verify CoTexT's performance on other CodeXGlue dataset operations such as code generation and error checking. These activities regularly lead to SOTA results, demonstrating the evolution of our model.

Shoebi et al. colleagues [10] proposed Megatron-LM: Training multi-billion parameter language models.

Building massive Transformer models can advance the state of the art in natural language processing applications, according to recent research on language modelling. However, memory constraints can make training very large datasets exceedingly challenging. This work uses a simple and effective parallel modelling technique that can train variable models with thousands of parameters. Methods for training large transformer models are also shown. This approach can be fully implemented by adding some communication to native PyTorch, is orthogonal and complementary to the comparison model, and does not require new modifications or library modifications. Using 512 GPUs to compile a model with up to 8.3 billion variables is not an example of this approach. Compared to a GPU core supporting 39 TeraFLOPs (30% of peak FLOPs), the entire application supports 15.1 PetaFLOPs with 76% scaling efficiency. To demonstrate how cutting-edge (SOTA) large language models can be, a Transformer language with 8.3 billion parameters (similar to GPT-2) and a model with 3.9 billion parameters (similar to BERT) were trained.

Yue Wang et al. [9] published CodeT5: A descriptor-aware combined pre-trained encoder-decoder model for code understanding and generation.

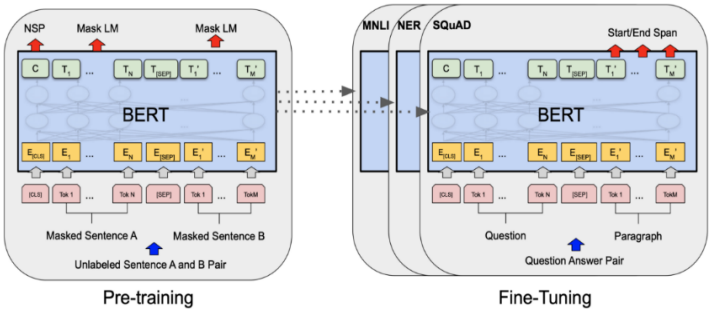
Recent research has shown that natural language (NL) pre-trained models such as BERT and GPT adapt well to programmatic language (PL) to assist with a variety of coding-related responsibilities. Despite their success, most existing approaches treat the sentence as NL, ignore special properties of the PL (such as token type), or rely solely on the work before generating the best encoder (or decoder only). comprehension) activities. CodeT5 is a pre-programmed encoder-decoder Transformer model that uses developer-generated identifiers to better leverage code semantics. The model enables multitask learning and uses a unified framework to support code interpretation and design tasks. The model can also identify which tokens are tokens and recover them when they are blocked, thanks to new feature-aware pre-match training. Extensive testing has shown that CodeT5 outperforms standard algorithms on tasks such as code recognition and clone detection, as well as tasks developed in multiple languages such as PL-NL, NL-PL, and PL-PL. Further research has shown that this model can extract semantic information from the process more efficiently.

I. SYSTEM ARCHITECHTURE

1. BERT is the core of the Transformer encoder layer, which contains multiple self-monitoring "heads".

2. For each input token in the queue, each head computes priority, value, and query vectors, which are used to create a weighted representation/control.

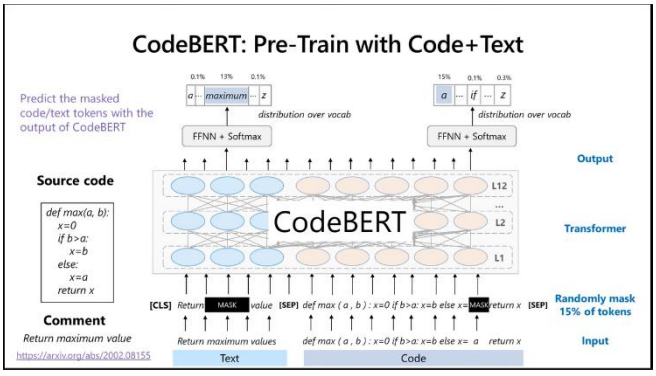
3. The results of all heads in the same layer are combined and passed through the entire bonding process. Each layer is wrapped with cross-links and then the layer is normalized.



4. The normal operation of BERT consists of two stages: pretraining and optimization.

5. BERT is easily extendable to multi-modality, i.e., training with different types of datasets. CodeBert is a bi-modal extension of Bert. i.e. CodeBERT utilizes both natural languages and source codes as its input. (Unlike, traditional BERT and Roberta focus on natural language primarily)

BERT is easily extendable to multi-modality, i.e., training with different types of datasets. CodeBert is a bi-modal extension of Bert. i.e. CodeBERT utilizes both natural languages and source codes as its input. (Unlike, traditional BERT and Roberta that focus on natural language primarily)

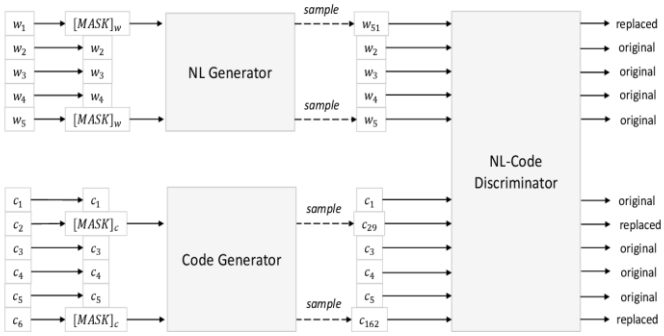


CodeBERT primarily describes two training objectives:

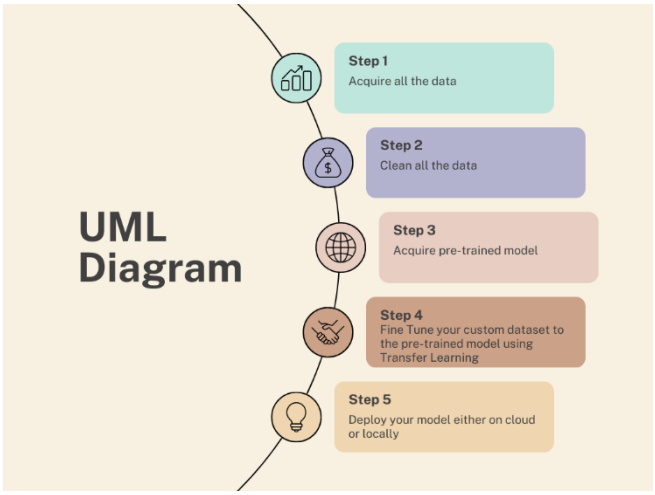
1) Use the masking language model (MLM) to train CodeBERT: to mask, select and bite different functions for

NL and PL, then use the [mask] custom Card command. The purpose of MLM is to predict the original tokens covered.

2) CodeBERT Tutorial using Replacement Token Detection (RTD): Here, in the first NL array and PL array, several tokens will be randomly covered. Then train the generator model, which is a probabilistic model with N. Training a discriminant model to determine whether a word is an obsolete word is a binary classification problem.



II. UML Diagram



III. Algorithm

1) **Kendall's Tau Correlation**

• The Kendall tau correlation, which compares the anticipated and ground truth cell ordering over the whole set of test set notebooks, is used to assess the predictions.
• Determine S, the quantity of neighboring entry swaps required to convert the predicted cell order into the ground truth cell order.
• A notebook with n cells will require, at most, 1/2 n (n − 1) swaps to sort a projected order.
• We total the number of swaps from your anticipated cell order for every notebook in the test set, and we do the same for the worst-case number of swaps.

• Formula for Kendall tau correlation:

$$K = 1 - 4\frac{\sum_i S_i}{\sum_i n_i(n_i - 1)}$$

IV. Result Screenshot, Tables and Analysis

After using the model to predict the test dataset, the following results are obtained. Note that, the id refers to the unique id of the notebook and the cell order specifies the predicted order of code-comment cells

id	cell_order
0009d135ece78d	0a226b6a ddfd239c 8cb8d28a c6cd22db 1372ae9b e25aa9bd 90ed07ab ba55e576 7f388a41 f9893819 2843a25a 39e937ec 06dbf8cf
0010483c12ba9b	7f270e34 54c7cab3 fe66203e 7844d5f8 5ce8863c 4a0777c4 4703bb6d 4a32c095 865ad516 02a0be6d
0010a919d60e4f	23607d04 b7578789 aafc3d23 bbff12d4 80e077ec 584f6568 89b1fdd2 b190ebb4 8ce62db4 ed415c3c d3f5c397 18ce8cc0 35cd0771 322850af 7f53de45 c069ed33 50bc28b3 5115ebe5 868c4eae 5e8c5e7e 1d4dbeae 4ae17669 3f4a105f 80433cf3 bac960d3 a4875f3f bd8fbd76 8a0842b8 0e2529e8 1345b8b2 ea06b4d0 52fe98c4 cdae286f 03cb1feb 724d27d3 4907b9ef 44eb815a d65238ba 3bff2378 7d157458 8679f842 641e45c1 83514fa3 7f6a2fa8 f9e38e5a b78215d1 e52e4a9e 982d964e 9f5d983e 22776759 ef01da10 e0bf4b8b 7317e652 5793f12e 3741e756 bc8eaa53 f7f2ce31 0115f7f5 21b6fb8f 177f908c 4356ab34 d2f722a5
0028856e09c5b7	eb293dfc 012c9d02 d22526d1 3ae7ece3

V. Advantages

1) The Transformer model represents a novel form of encoder-decoder architecture that leverages self-awareness to comprehend sequences of speech.
2) This capability enables parallel processing, resulting in significantly faster performance compared to other models of similar efficacy.
3) The potential of Transformers extends to understanding the connections between sequential elements even when they are widely spaced.
4) Transformers exhibit heightened accuracy in their operations.
5) These models exhibit an egalitarian approach, distributing equal attention to all elements within a given sequence.
6) Transformers demonstrate the ability to process and train on larger datasets within shorter time frames.
7) Their versatility allows Transformers to effectively work with diverse types of sequential data.
8) Transformers prove advantageous in anomaly detection applications.

VI. Disadvantages

1) The base key uses O(n^2) in memory/computation. This is why anything over 1000 steps is difficult to study outside of a dedicated centre.
2) As the model becomes more efficient, the bias of the language model increases.

VII. Applications

1. Code Understanding
2. Code Generation
3. Code Translation
4. Code Localization/Refactoring/Labelling
5. Automatic Comment Generation
6. Code Question Answering
7. Code Natural Language Search

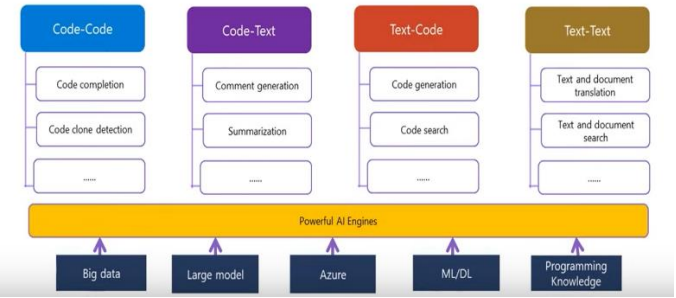
8. Bug/Fault/Defect Prediction
9. Automatic Test Case Generation
10. Automatic Code Summarization/Documentation
11. Testing/Maintenance Effort Prediction
12. Code Suggestions/IntelliSense

1. Code-to-Code Translation: This can be used for full code or code translation. For example, when a developer wants to write Java code that already includes Python code, the code definition language can help define that block of code.

2. Converting program code to text: can help developers express program code. For example, when a developer looks at an unknown piece of code, the CodeAI model can translate the code into natural language and summarize the content for the developer.

3. Text Code: This provides similar functionality to searching for a code. This search helps users retrieve the relevant code based on the query.

4. Text to Text: can help translate text into multiple languages.



VIII. Future Scope

1. Other creative techniques including non-machine-learning-based approaches aimed at better understanding the relationship between comment cells and code cells can be explored
2. In the future, we can add multi-task learning capabilities to our model and use them for various other software engineering tasks.
3. We can also apply sparsification or pruning techniques to reduce the size of our model.
4. We can add ExplainableAI and SafeML technologies to our model to make it more transparent and ethical.

IX. Conclusion

We have developed a model capable of understanding and comprehending natural language as well as code. This model can predict, correctly, the natural language comments for each code block.

X. Abbreviations

1) NLP: Natural Language Processing
2) GPT3: Generative Pre-trained Transformer 3
3) LM: Language Modelling
4) RTD: Replaced Token Detection
5) coTEXT: Code-Text Transformer
6) SOTA: state-of-the-art
7) DNN: Deep Neural Network

Acknowledgement

I would like to take this opportunity to thank my great mentor, Dr. K. S. Thakre, for providing me with the wonderful chance to choose and give this seminar as well as the resources I needed to finish it successfully. I would like to express my gratitude to Dr. K. S. Thakre, Head of the Department of Computer Engineering, for opening the department to the realization of the seminar, as well as to all the staff members for their invaluable assistance, priceless recommendations, and extremely valuable time provided when needed. I would like to express my gratitude and respect to everyone who has assisted me, whether directly or indirectly.

References

[1] Allamanis, Miltiadis and Barr, Earl T and Devanbu, Premkumar and Sutton, Charles, "A survey of machine learning for big code and naturalness", ACM Computing Surveys (CSUR), Vol. 51, 5 May 2018.

[2] Feng, Zhangyin and Guo, Daya and Tang, Duyu and Duan, Nan and Feng, Xiaocheng and Gong, Ming and Shou, Linjun and Qin, Bing and Liu, Ting and Jiang, Daxin and Zhou, Ming, CodeBERT: A Pre-Trained Model for Programming and Natural Languages, arXiv, 18 Sep 2020

[3] Phan, Long and Tran, Hieu and Le, Daniel and Nguyen, Hieu and Anibal, James and Peltekian, Alec and Ye, Yanfang, CoTexT: Multi-task Learning with Code-Text Transformer, arXiv, 21 Jun 2021

[4] Ahmed, Toufique and Devanbu, Premkumar, Learning code summarization from a small and local dataset, arXiv,2 Jun 2022

[5] Garg, Spandan and Moghaddam, Roshanak Zilouchian and Clement, Colin B. and Sundaresan, Neel and Wu, Chen, DeepPERF: A Deep Learning-Based Approach For Improving Software Performance, arXiv, 27 Jun 2022

[6] Xu, Frank F. and Alon, Uri and Neubig, Graham and Hellendoorn, Vincent J., A Systematic Evaluation of Large Language Models of Code, arXiv, 4 May 2022

[7] Daya Guo1∗ , Shuo Ren2∗ , Shuai Lu3∗ , Zhangyin Feng4∗ , Duyu Tang5 , Shujie Liu5 , Long Zhou5 , Nan Duan5 , Alexey Svyatkovskiy6 , Shengyu Fu6 , Michele Tufano6 , Shao Kun Deng6 , Colin Clement6 , Dawn Drain6 , Neel Sundaresan6 , Jian Yin1 , Daxin Jiang7 , and Ming Zhou5 1School of Computer Science and Engineering, Sun Yat-sen University. 2Beihang University, 3Peking University, 4Harbin Institute of Technology, 5Microsoft Research Asia, 6Microsoft Devdiv, 7Microsoft STCA, GRAPHCODEBERT: PRE-TRAINING CODE REPRESENTATIONS WITH DATA FLOW, ICLR, 2021

[8] Peng, Dinglan and Zheng, Shuxin and Li, Yatao and Ke, Guolin and He, Di and Liu, Tie-Yan, How could Neural Networks understand Programs?, arXiv, 31 May 2021

[9] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi, CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation, arXiv, 2 Sep 2021

[10] Shoeybi, Mohammad and Patwary, Mostofa and Puri, Raul and LeGresley, Patrick and Casper, Jared and Catanzaro, Bryan, Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism, arXiv, 13 Mar 2020