Detection of Code Clones using BERT Models

Akash Kumar Pujari
Asrith Raghavendra Madanala
Siddhaarth Mandava
akashpujari@vt.edu
asrithraghavendr@vt.edu
siddhaarthm1097@vt.edu

Abstract

Code clone detection is an essential issue for software maintenance and development. The term "code clone" refers to a pair of semantically related code fragments that are present in the code base. These parallels come in four different varieties, each of which is described in depth in the introduction. Clones of code are created by programmers when they create software. To increase programming efficiency inside the same organization, developers duplicate existing code with or without modification. By requiring the same or identical bug fixes or program changes to be implemented uniformly across numerous code places, these code clones may make software maintenance more difficult. The development and maintenance of software systems will be negatively impacted by excessive code clones. Consequently, identifying code clones is a significant issue in the software industry. The identification of code plagiarism is another area where our project may be helpful.
In this project, we worked on measuring the semantic similarity between codes using the BigCloneBench dataset, part of the CodeXGLUE benchmark dataset. This dataset mainly consists of Java code. For our project, we worked on finetuning three different models - CodeBERT, RoBERTa and DistilRoBERTa. We have also developed a web application with our CodeBERT model integrated into it, where users may enter two different sets of code and the model will determine whether or not they are identical. The main motivation behind choosing these three variants of BERT were

1. Comparing the performance between the bimodal CodeBERT and unimodal RoBERTa and DistilRoBERTa models on code clone detection on BigCloneBench dataset.
2. Checking the performance of 40% less expensive and 60% faster DistilRoBERTa with Roberta on code clone detection.
3. Evaluating the impact of pre-training on software domains (CodeBERT) on the model performance when comparing it to the performance of a model not pre-trained specifically in this domain.

Further, we tried to extract insights from result analysis to improve and fine-tune the models. Analysis was also conducted on which type of clone pairs the models are failing to predict and what changes could be incorporated to improve them. The final analysis of the study can be found in the analysis section. The performance metric that we used used is F-1 score along with precision and recall. We obtained the best results for the CodeBERT model, in line with what we expected. The DistilRoBERTa model outperformed RoBERTa on this dataset for clone detection. Detailed results can be found in the results section of our report.

Keywords: Code clone detection, BERT, CodeBERT, BigCloneBench, CodeXGlue, RoBERTa, DistilRoBERTa

1 Introduction

The method of determining whether two sets of codes are similar or not (i.e., clones) is known as code clone detection. The necessity for an architecture or model to determine whether two pieces of code are clones has made code cloning a major area of research in recent years due to the growth in the quantity of code used in today's world.
This might be to check plagiarism, increase code

re-usability across the same organization etc. Code Clones generally fall into 4 different categories:

1. Code blocks which are exact copies of each other with only changes in white-spaces and comments.
2. Code blocks which are identical, but the variable names are changed across both the blocks.
3. Similar codes which differ at statement level.
4. Code blocks with different statements but implementing the same functionality.

In this project, we worked to enable different variants of the BERT model to perform well on code clone detection. The models which were used are CodeBERT, RoBERTa and DistilRoBERTa. The main motivation behind this project was to compare all the above-mentioned models and conduct a thorough analysis of all of them to improve the models. We also aimed to evaluate the impact of pre-training on software domains (CodeBERT) on the model performance when comparing it to the performance of a model not pre-trained specifically in this domain. We compared the performances of these two models with the CodeBERT model performance. We also created a web application which integrated our CodeBERT model into it where two sets of code can be inputted into the website and the model will detect whether they are clones or not.

For Code Cloning tasks, there are some standard datasets. The dataset that we have used is BigCloneBench [11] within CodeXGLUE [7]. This project specifically works on binary classification (0/1). Input given was two codes and the result was in the form of (0/1) , where 1 stands for semantic equivalence or cloned code and 0 for unique codes.

2 Related Work

There has been research work conducted in this area with different proposed methods leading to varying results. Some initial work include line-by-line matching for an abstracted source program [1] and similarity detection for metrics values of function bodies [8]. In 1999, A language dependent clone detection tool Duploc [2] was proposed which used string-based Dynamic Pattern Matching (DPM) to detect clones. The computation complexity was O(n^2) for the input size n and it was practically too expensive. Kamiya T. et. al in a paper published in 2002, proposed a new clone detection technique which consisted of the transformation of input source text and a token-by-token comparison [5]. They developed a tool called CCFinder (Code Clone Finder), which extracts code clones in C, C++, Java, COBOL and other source files. Yoshiki Higo et. al. proposed an incremental PDG-based approach for code clone detection whose advantage was that it could detect non-contiguous code clones [4].
In 2016, Martin White et. al introduced a framework, which relies on deep learning, for automatically linking patterns mined at the lexical level with patterns mined at the syntactic level [13] and got good results with it. Last year (2021), Kun Xu and Yan Liu published a paper in which they proposed SCCD-GAN, an enhanced semantic code clone detection model which is based on a graph representation form of programs and uses Graph Attention Network to measure the similarity of code pairs [14]. It achieved a lower detection FPR than existing methods.

3 Dataset

The dataset that was used is BigCloneBench which is a part of the CodeXGLUE dataset. CodeXGLUE is a benchmark dataset by microsoft which contains 14 datasets for 10 different code Intelligence tasks. The main motivation behind choosing this dataset is that it has a sample size of around 1.7M. Transformer models are always data hungry models and generalize best when we have more data samples. Moreover, on analyzing the dataset, we found that the BigCloneBench training set was highly balanced. We observed that the training set consisted of 450862 positive labels and 450166 negative labels. This is a distribution of 50.03% to 49.96% which is quite balanced. Therefore, we decided to use the CodeXGLUE dataset. The structure of CodeXGLUE dataset is as follows:

1. The data is stored in jsonlines format. Each line in the uncompressed file represents one function in Java. A sample row consists of:
   a. func: the function
   b. idx: index of the example
2. The train, validation and testing files provide examples, stored in the following format: idx1 idx2 label

The split used for training validation and testing was approximately 50-25-25 since we have a large enough sample size.

Table 1. Distribution of examples

Train	901,028
Validation	415,416
Test	415,416

4 Models and Training

General Architecture

We used three different models for our project - CodeBERT, RoBERTa and DistilRoBERTa.
The encoder used was based off Roberta with different number of layers based on the model used - 12 layers in CodeBERT and RoBERTa and 6 layers in DistilRoBERTa.
We used a RoBERTa classification head with three layers:

1. A dense layer with 1536 input features and 768 output features.
2. A dropout layer with probability of 0.1.
3. An output layer with 768 input features and 2 output features (one for each label).

Following this, a softmax activation function was used to return the probabilities.

CodeBERT

CodeBERT [3] is a model specifically designed for NL-PL downstream tasks. It is a bimodal pre-trained model having the same architecture as the RoBERTa base with 125M parameters. The original pre-trained CodeBERT model is trained on both bimodal as well as unimodal training data. Training data that was used for CodeBERT was generated from open source GitHub repositories.
The input that we used for training was of the form: index of java code1, index of java code 2, class label. The input was then broken down into different batches. The average input length for java code in the dataset was 1579 which is way greater than the CodeBERT model max length limit which is 512 but since the median input length was around 405 we set the max length for the model as 400.

The hyper-parameters that were used to train the CodeBERT model are shown in Table 2:

Figure 1. Transformer Model Architecture [12]

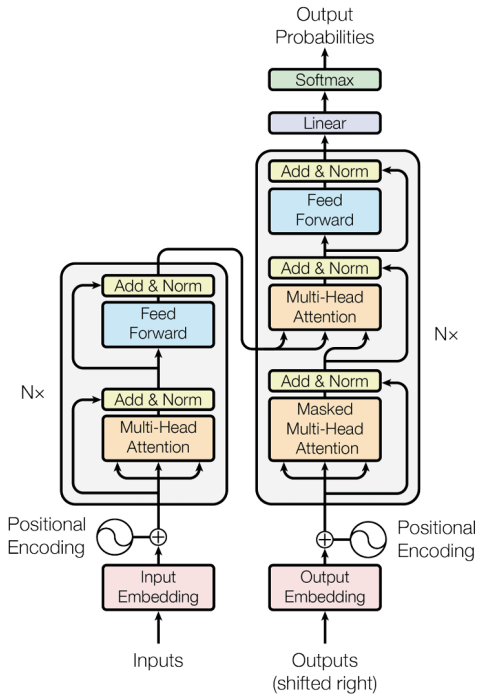
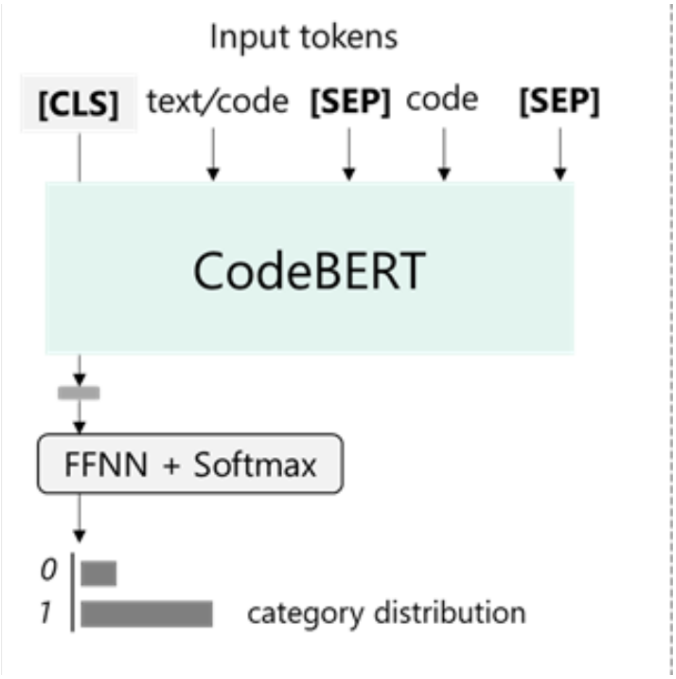


Table 2. Hyperparameters used to train CodeBERT

S.no	Hyperparameter	Value
1	Train batch size	8
2	Evaluation batch size	16
3	Epochs	2
4	Learning Rate	3e-5
5	Optimizer	Adam
6	Loss	CrossEntropyLoss
7	Dropout	0.1
8	Max Sequence Length	400

Figure 2. CodeBERT Pipeline [9]



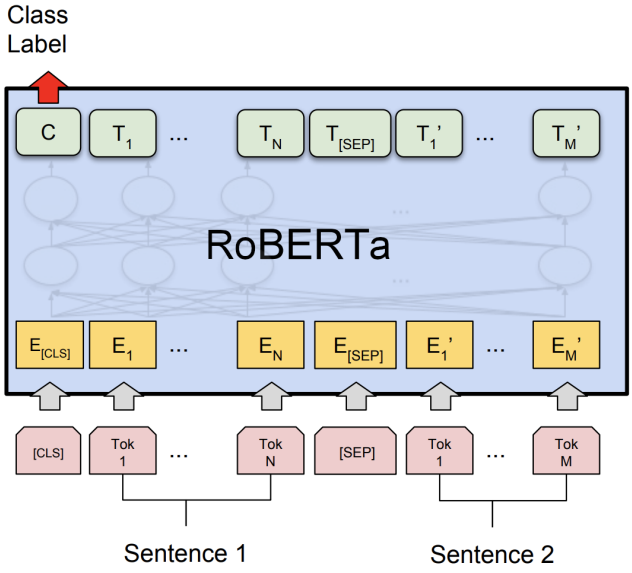
The optimizer we used was the Adam optimizer with a Cross Entropy loss function. We got good results after training the model for 2 epochs. The results were recorded and then analyzed to find exactly where the model is making mistakes. We found that most of the wrong predictions by the model were for type-4 code clone pairs. A detailed analysis can be found in the analysis section of our report.

RoBERTa

The RoBERTa model proposed an improved recipe for training BERT models, that can match or exceed the performance of all of the post-BERT methods. The modifications to existing BERT implementations include [6]:

1. Training the model longer, with bigger batches, over more data
2. Removing the next sentence prediction objective
3. Training on longer sequences
4. Dynamically changing the masking pattern applied to the training data.

Figure 3. RoBERTa Architecture [6]



The RoBERTa model we used has 12-layers, 12-heads and a total of 125M parameters.
We wanted to see how a non bimodal model that has not been pretrained on software PL tasks would perform on a code clone detection task, and look at the effect that pretraining has when comparing the results obtained by RoBERTa to results obtained by CodeBERT.
The hyperparameters used for training the RoBERTa model are shown in Table 3:

Table 3. Hyperparameters used to train RoBERTa

S.no	Hyperparameter	Value
1	Train batch size	8
2	Evaluation batch size	16
3	Epochs	2
4	Learning Rate	3e-5
5	Optimizer	Adam
6	Loss	CrossEntropyLoss
7	Dropout	0.1
8	Max Sequence Length	400

DistilRoBERTa

The DistilRoBERTa model distilled version of the RoBERTa model. It has 6 layers, 768 dimension and 12 heads, totalizing 82M parameters (compared

to 125M parameters for RoBERTa). On average DistilRoBERTa is twice as fast as Roberta-base. The model was pre-trained with three objectives [10]:

1. Distillation loss: the model was trained to return the same probabilities as the RoBERTa base model
2. Masked language modeling (MLM): this is part of the original training loss of the RoBERTa base model. When taking a sentence, the model randomly masks 15% of the words in the input then run the entire masked sentence through the model and has to predict the masked words.
3. Cosine embedding loss: the model was also trained to generate hidden states as close as possible as the RoBERTa base model.

This way, the model learns the same inner representation of the English language than its teacher model, while being faster for inference or downstream tasks. In this project, we wanted to look at the effect of distillation to the performance of the model in the code domain, and hence compare the performance of RoBERTa and DistilRoBERTa on code clone detection and then compare their performance to CodeBERT which has been pre-trained in this domain.
The hyperparameters used for training DistilRoBERTa are shown in 4:

Table 4. Hyperparameters used to train DistilRoBERTa

S.no	Hyperparameter	Value
1	Train batch size	8
2	Evaluation batch size	16
3	Epochs	2
4	Learning Rate	3e-5
5	Optimizer	Adam
6	Loss	CrossEntropyLoss
7	Dropout	0.1
8	Max Sequence Length	400

5 Evaluation

Once the model is trained, it is important to validate the model's ability to generalize on unseen data in order to validate that the model has actually learned something and it has not overfit to the training data.
For testing the performance of our trained models, we passed a list of pairs of indexes, where these indexes represented the positions of the code in the jsonlines data file. Once we passed these pairs of indexes, we had the model predict whether the two codes were clones or not and stored these labels along with the index pair in a predictions.txt file.
We then compared the labels predicted by the model to the actual labels and calculated the precision, recall and F1-score.
Recall measures among all known true clone pairs, how many of them are detected by a clone detection approach.

$$Recall = \frac{No.\ of\ true\ clone\ pairs\ detected}{Total\ No.\ of\ known\ true\ clone\ pairs}$$

Precision measures among all of the clone pairs reported by a clone detection approach, how many of them are actually true clone pairs.

$$Precision = \frac{No.\ of\ true\ clone\ pairs}{Total\ No.\ of\ detected\ clone\ pairs}$$

F1-score is the harmonic mean of precision and recall.

$$\text{F1-score} = \frac{2 * precision * recall}{precision + recall}$$

6 Results

The training, validation and testing results that we obtained are shown in tables 5, 6 and 7 respectively. We have used precision, recall and F-score as the evaluation metrics.

Table 5. Training Scores

Model	Precision	Recall	F Score
CodeBERT	0.9261	0.9353	0.9307
RoBERTa	0.877	0.9656	0.9196
DistilRoBERTa	0.9206	0.931	0.9258

Table 6. Validation Scores

Model	Precision	Recall	F Score
CodeBERT	0.9289	0.9464	0.9376
RoBERTa	0.9103	0.9286	0.9194
DistilRoBERTa	0.9112	0.9311	0.9210

Table 7. Testing Scores

Model	Precision	Recall	F Score
CodeBERT	0.9341	0.9567	0.9453
RoBERTa	0.9041	0.9360	0.9198
DistilRoBERTa	0.9186	0.9370	0.9277

7 Analysis

The results that we got were as expected with CodeBERT giving us the best F1-score. After further analyzing where exactly CodeBERT is making mistakes, we found that out of the 6842 pairs that our model predicted wrong, 86% of the time, the model actually predicted 0 when the ground truth was 1 signifying that the model failed to account for data cloning between the two input codes in these cases.

To understand why the model was predicting the incorrect label for the code combination, we examined the code pairings where it failed. We discovered that the majority of these occurrences fall under type-4 code cloning. Type-4 clones are copies where both code blocks are designed to produce the same result. Deeper investigation revealed that these codes' input length exceeded their maximum length. Therefore, it was probable that the comparable code blocks were actually longer than the model's 400 maximum length.

It is also possible that the CodeBERT is not trained on that type of code in its pre-training stages and therefore failed to realize that in fact these two code blocks will give the same result. We received an unexpected result for RoBERTa and DistilRoBERTa. Due to its more complex structure and greater number of factors, we had anticipated RoBERTa to perform better than DistilRoBERTa, however our experimental findings indicate otherwise.

Out of all the wrong predictions that the model made, for 14% of the predictions, the model predicted the code pair to be clones 14% of the time when they were not. Even though the number is not huge, it can hurt the system in longer runs.

8 Conclusion

Through our project, We aimed to answer three main research questions:

1. Comparing the performance between the bimodal CodeBERT and unimodal RoBERTa and DistilRoBERTa models on code clone detection on BigCloneBench dataset.
2. Checking the performance of 40% less expensive and 60% faster DistilRoBERTa with Roberta on code clone detection.
3. Evaluating the impact of pre-training on software domains (CodeBERT) on the model performance when comparing it to the performance of a model not pre-trained specifically in this domain.

We found the following results:

Bimodality of CodeBERT helps the model to learn the NL-PL embeddings better and therefore according to results that we got, we can conclude indeed CodeBERT is outperforming RoBERTa and DistilRoBERTa as per the expectations.

DistilRoBERTa is actually performing better than RoBERTa. DistilRoBERTa is computationally very cheap as compared to RoBERTa and also it is faster. Therefore, for some reason if we are not using the first choice CodeBERT for NL-PL tasks, DistilRoBERTa is not at all a bad option. The performance of a model that has been pre-trained in the software domain with codes improves the model performance by 2-3% with respect to the F-scores obtained.

References

[1] Brenda S. Baker. 1992. A program for identifying duplicated code. *Computing Science and Statistics* (1992).

[2] Stéphane Ducasse, Matthias Rieger, and Serge Demeyer. 1999. A Language Independent Approach for Detecting Duplicated Code. IEEE, 109–118.

[3] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin,

Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. http://arxiv.org/abs/2002.08155 cite arxiv:2002.08155Comment: Accepted to Findings of EMNLP 2020. 12 pages.

[4] Yoshiki Higo, Ueda Yasushi, Minoru Nishino, and Shinji Kusumoto. 2011. Incremental code clone detection: A PDG-based approach. In *2011 18th Working Conference on Reverse Engineering*. IEEE Computer Society, 3–12.

[5] T. Kamiya, S. Kusumoto, and K. Inoue. 2002. CCFinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering* 28, 7 (2002), 654–670. https://doi.org/10.1109/TSE.2002.1019480

[6] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. https://doi.org/10.48550/ARXIV.1907.11692

[7] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. https://doi.org/10.48550/ARXIV.2102.04664

[8] Jean Mayrand, Claude Leblanc, and Ettore Merlo. 1996. Experiment on the automatic detection of function clones in a software system using metrics. *1996 Proceedings of International Conference on Software Maintenance* (1996), 244–253.

[9] Microsoft. 2020. CodeXGLUE: A benchmark dataset and open challenge for code intelligence. https://www.microsoft.com/en-us/research/blog/codexglue-a-benchmark-dataset-and-open-challenge-for-code-intelligence/.

[10] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *ArXiv* abs/1910.01108 (2019).

[11] Jeffrey Svajlenko, Judith F. Islam, Iman Keivanloo, Chanchal K. Roy, and Mohammad Mamun Mia. 2014. Towards a Big Data Curated Benchmark of Inter-project Code Clones. In *2014 IEEE International Conference on Software Maintenance and Evolution*. 476–480. https://doi.org/10.1109/ICSME.2014.77

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. https://doi.org/10.48550/ARXIV.1706.03762

[13] Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk. 2016. Deep learning code fragments for code clone detection. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 87–98.

[14] Kun Xu and Yan Liu. 2021. SCCD-GAN: An Enhanced Semantic Code Clone Detection Model Using GAN. In *2021 IEEE 4th International Conference on Electronics and Communication Engineering (ICECE)*. 16–22. https://doi.org/10.1109/ICECE54449.2021.9674552