# TreeBERT: A Tree-Based Pre-Trained Model for Programming Language Supplementary Material

**Xue Jiang**[1] **Zhuoran Zheng**[2] **Chen Lyu**[*1] **Liang Li**[1] **Lei Lv**[1]

[1]School of Information Science and Engineering, Shandong Normal University, China
[2]School of Computer Science and Engineering, Nanjing University of Science and Technology, China

In this supplemental material, we first introduce the code tokenization in Section 1. Second, we provide detailed statistical information of datasets used for the experiment in Section 2. Then, we describe the metrics used to evaluate TreeBERT in Section 3. Finally, we show the detailed results of some experiments in Section 4.

## 1 CODE TOKENIZATION

Due to the strong structure of code, indentation is meaningful in Python, which cannot be removed simply by splitting code. Follow [Rozière et al., 2020], we use "INDENT" and "DEDENT" instead of indentation to indicate the beginning and end of a block of code. "NEWLINE" is used to represent line breaks. Spaces are replaced with "_" in strings, and code comments are removed. An example of a processed Python code snippet is shown in Figure 1.

**Python code snippet**

```
import sys
from os.path import dirname, join as join_path

def sys_path  () :
    """ Add `./third_party` to `sys.path`.
    """

    third_party_dir = join_path(dirname(__file__), 'third party')
    if not third_party_dir in sys.path:
        sys.path.insert(1, third_party_dir)
```

```
import sys NEWLINE from os . path import dirname , join as join_path NEWLINE
def sys_path ( ) : NEWLINE INDENT third_party_dir = join_path ( dirname ( __
file __ ) , ' third_party ' ) NEWLINE  if not third_party_dir in sys . path :
NEWLINE INDENT sys . path . insert ( 1 , third_party_dir ) DEDENT DEDENT
```

Figure 1: Example of code tokenization.

## 2 DATA STATISTICS

Table 1 shows detailed statistics of the four datasets used for code summarization, namely, ETH Py150[1], Java-small[2], Java-med[3], and Java-large[4]. Table 2 shows detailed statistics for two datasets, a Java dataset[5] from DeepCom [Hu et al., 2018] for code documentation and a C# dataset[6] from CodeNN [Iyer et al., 2016] for evaluating the performance of the model on pre-training unseen language.

Table 1: Statistics of datasets used for code summarization.

| | ETH Py150 | Java-small | Java-med | Java-large |
|---|---|---|---|---|
| Example Number(train) | 143,310 | 665,115 | 3,004,536 | 15,344,512 |
| Example Number(valid) | 33,878 | 23,505 | 410,699 | 320,866 |
| Example Number(test) | 35,714 | 56,165 | 411,751 | 417,003 |
| Avg.number of Paths(train) | 130 | 171 | 187 | 220 |
| Avg.path length(train) | 19 | 21 | 23 | 22 |
| Avg.comments length(train) | 3 | 3 | 3 | 3 |

## 3 EVALUATION METRICS

In this section, we provide details of the calculation of precision, recall, and F1 score used in the code summarization and BLEU used in code documentation.

**Precision, Recall, F1-Score** In code summarization, we do not use accuracy and BLEU since the generated func-

[*]Corresponding author (lvchen@sdnu.edu.cn)

[1]https://www.sri.inf.ethz.ch/py150
[2]https://s3.amazonaws.com/code2seq/datasets/java-small.tar.gz
[3]https://s3.amazonaws.com/code2seq/datasets/java-med.tar.gz
[4]https://s3.amazonaws.com/code2seq/datasets/java-large.tar.gz
[5]https://github.com/xing-hu/DeepCom/blob/master/data.7z
[6]https://github.com/sriniiyer/codenn/tree/master/data/stackoverflow/csharp

Table 2: Statistics for DeepCom's Java dataset and CodeNN's C# dataset.

| | Java | C# |
|---|---|---|
| Example Number(train) | 450,124 | 52,812 |
| Example Number(valid) | 55,310 | 6,601 |
| Example Number(test) | 54,871 | 6,602 |
| Avg.number of Paths(train) | 212 | 207 |
| Avg.path length(train) | 19 | 16 |
| Avg.comments length(train) | 12 | 10 |

tion names are composed of subtokens and are relatively short (average length of 3 subtokens). Following Alon et al. [2019b,a]., we use precision, recall, and F1 as metrics. The calculation is as follows.

$$Precision = \frac{TP}{TP+FP}$$
$$Recall = \frac{TP}{FN}$$
$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

When the predicted subtoken is in the function name, we treat it as a true positive ($TP$). When the predicted subtoken is not in the function name, we treat it as a false positive ($FP$). When the subtoken is in the function name but is not predicted, we treat it as a false negative ($FN$). The label "$UNK$" is counted as $FN$; thus, the prediction of this word will reduce the recall value.

**BLEU** The BLEU score can be used to measure the similarity between the generated comments and the reference code comments at the token level, and it is calculated as follows.

$$BLEU = BP \cdot exp\left(\sum_{n=1}^{N} w_n \cdot logp^n\right)$$
$$BP = \begin{cases} 1, & c > r, \\ e^{1-r/c}, & c \leq r. \end{cases}$$

where the upper limit of $N$ is taken as 4, i.e., at most 4-grams are computed, $w_n = \frac{1}{N}$, and $p_n$ is ratio of the clauses of length $n$ in the candidate to those also in the reference.

In brevity penalty (BP), $r$ denotes the length of the reference annotation and $c$ denotes the length of the annotation generated by the model.

## 4 MORE EXPERIMENTAL RESULTS

Figure 2 shows the visualization results of the F1 score of code summarization. Table 3 gives the detailed results of the ablation study.

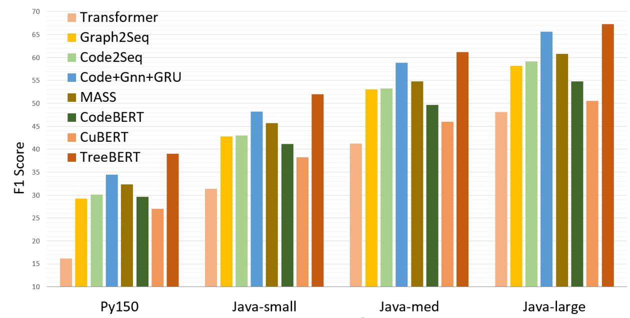


Figure 2: code summarization visualization results for F1 scores on different datasets.

Table 3: Results of the ablation study.

| Model | BLEU | ΔBLEU |
|---|---|---|
| TreeBERT | 20.49 | - |
| No PMLM | 14.12 | -6.37 |
| No NOP | 16.71 | -3.78 |
| No Node Position Embedding | 20.25 | -0.24 |
| Randomly Masking Nodes | 14.81 | -5.68 |
| Only Masking Value Nodes | 18.25 | -2.24 |

## References


Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. code2seq: Generating sequences from structured representations of code. In *7th International Conference on Learning Representations, ICLR*, 2019a.

Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. code2vec: learning distributed representations of code. *Proc. ACM Program. Lang.*, 3:40:1–40:29, 2019b.

Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. Deep code comment generation. In *Proceedings of the 26th Conference on Program Comprehension, ICPC*, 2018.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL*, 2016.

Baptiste Rozière, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. Unsupervised translation of programming languages. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems, NeurIPS*, 2020.