CAT-probing

A Metric-based Approach to Interpret How Pre-trained Models for Programming Language Attend Code Structure

Nuo Chen*, Qiushi Sun*, Renyu Zhu*, Xiang Li†, Xuesong Lu and Ming Gao

{nuochen, qiushisun, renyuzhu}@stu.ecnu.edu.cn
{xli, xslu, mgao}@dase.ecnu.edu.cn

East China Normal University
School of Data Science and Engineering

31 October 2022

Outline

Backgrounds

Fig 1. Pre-trained language models

Pre-trained language models have advanced the state-of-the-art across a series of NLP tasks. The success of these models for NL(Natural Language) leads to their application in the PL(Programming Language) domain.

Pre-trained Language Models for Code

Models	Inputs	Pre-training Tasks	Training Mode
RoBERTa	Natural Language(NL)	Masked Language Modeling(MLM)	Encoder-only
CodeBERT	NL-PL Pairs	MLM+Replaced Token Detection(RTD)	Encoder-only
GraphCodeBERT	NL-PL Pairs & AST	MLM+Edge Prediction+Node Alignment	Encoder-only
UniXcoder	NL-PL Pairs & Flattened AST	MLM ULM(Unidirectional Language Modeling) Denoising Objective(DNS)	Encoder & Decoder & Encoder-decoder

Table 1. The comparison of different language models mentioned in this paper.

What leads to CodePTMs' success?

CodePTMs perform quite well on downstream tasks

- How can they achieve such stunning performance?
- Beyond text information, do these models learn structure information?
- Do these models focus on the same points for different programming languages?

Thus, From the perspective of code structures, **Can these models capture the programming language's structure information?**

CAT-probing

Prior works

- Probing methods migrated from NLP
- Syntactic and semantic probing

CAT-probing

- One step forward, quantitatively evaluate how **C**odePTMs' **A**ttention scores relate to distances between AS**T** (Abstract Syntax Tree) nodes.

CAT-probing: U-AST

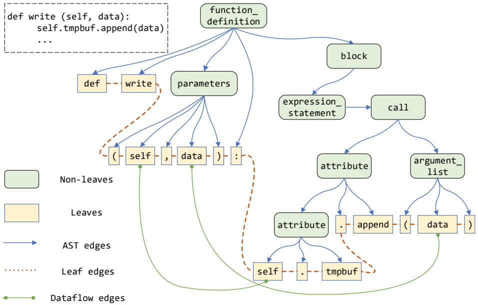


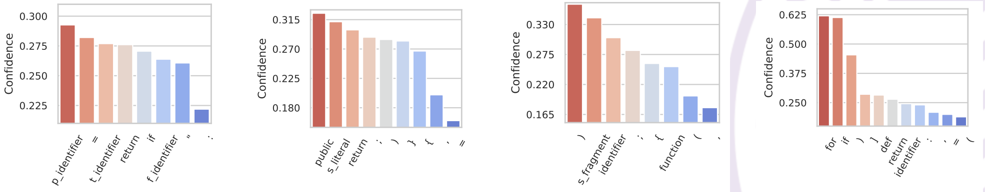
Fig 2. U-AST

What is U-AST?

- Based on abstract syntax tree (AST)
- Connect adjacent leaf nodes (Data flow edges)
- Increases AST's connectivity

Frequent Token Types

Language-specific frequent token types for four Promgramming languages.

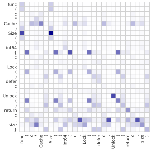
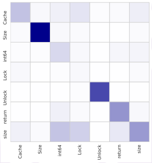


(a) Go (b) Java (c) JavaScript (d) Python

Fig 3. Visualization of the frequent token types on four programming languages.

CAT-probing: Token Selection

Source Code	Attention Heatmap	Attention Heatmap with Token Type Selection

`func (c *Cache) Size() int64 {`
`        c.Lock()`
`        defer c.Unlock()`
`        return c.size`
`}`
		

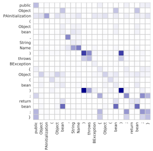
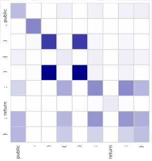
`class Solution {`
`    public Object postProcessAfterInitialization(Object bean, String beanName)`
`    throws BeansException {`
`        registerObject(bean);`
`        return bean;`
`    }`
`}`
		

Table 2. Heatmaps of the averaged attention weights in the last layer before and after using token selection, including Go and Java code snippets (from top to bottom).

CAT-probing: Code Matrices

- **Attention Matrix:** Constructed by token level attention scores.
- **Distance Matrix**: leaf nodes' distance of U-AST, Computed by shortest-path length.

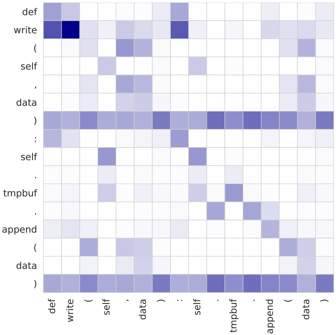


Fig 4. Attention Matrix

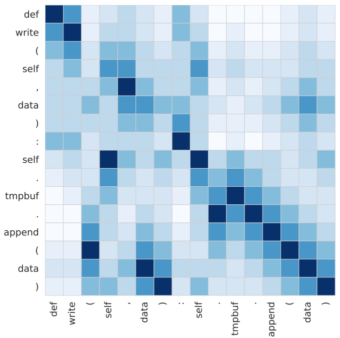


Fig 5. Distance Matrix

CAT-probing: CAT Score

A metric is designed to measure the capability of CodePTMs to attend code structure.

$$\text{CAT-score} = \frac{\sum_{C}\sum_{i=1}^{n}\sum_{j=1}^{n}\mathbb{1}_{\mathbf{A}_{ij}>\theta_A \text{ and } \mathbf{D}_{ij}<\theta_D}}{\sum_{C}\sum_{i=1}^{n}\sum_{j=1}^{n}\mathbb{1}_{\mathbf{A}_{ij}>\theta_A \text{ or } \mathbf{D}_{ij}<\theta_D}}, \tag{1}$$

The CAT-score and the CodePTMs' capability of attending code structure should be positively correlated

CAT-probing: Task

Code Summarization

- Comprehend code
- Automatically generate descriptions

```
def bubbleSort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0,n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
```

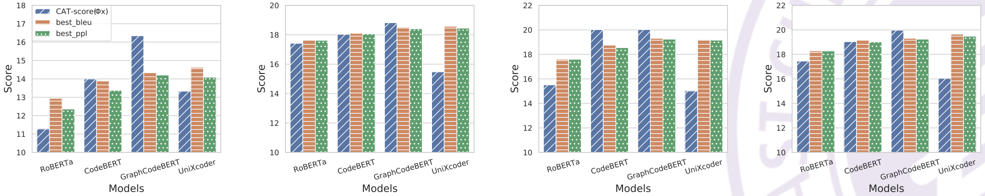
Sort the input array

Fig 6. Code Summarization

One of the most essential tasks of code representation learning

CAT-probing Effectiveness

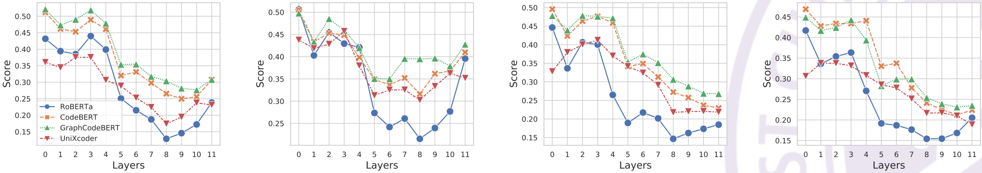
Comparison: CAT-scores and the models' performance



(a) JavaScript (b) Go (c) Java (d) Python

Fig 7. Comparisons between the CAT-score and the performance on code summarization task.

Layer-wise CAT-score



(a) Java (b) Go (c) JavaScript (d) Python

Fig 8. Layer-wise CAT-score results.

Layer-wise CAT-score Cont'd

1. As the layers increase, the CAT-scores decrease: some "special" tokens draw attention.
2. The relative magnitude relationship (GraphCodeBERT > CodeBERT > RoBERTa) between CAT-score is almost determined on all the layers and PLs.
3. Changes
   - Drastic change in middle layers, which are essential for transferring knowledge
   - In the last layers, CAT-scores gradually converge

Conclusion

- We proposed a novel probing method that can quantify the CodePTMs' ability to capture structural information.
- Experiments confirmed the feasibility of probing via attention distribution and code structure.
- Through CAT-probing, we obtained some interesting conclusions.

Limitation & Future works

Limitation

- Mainly focus on encoder-only CodePTMs
- Cannot completely avoid manual setting of hyperparameters

Future works

- Extend this probing method to more CodePTMs
- Create a unified probing method for different downstream tasks
- Design more general score functions

Thank You!