

Structure-aware Fine-tuning for Code Pre-trained Models

Jiayi Wu¹, Renyu Zhu³, Nuo Chen¹, Qiushi Sun⁴, Xiang Li¹, Ming Gao^{1,2*}

¹School of Data Science and Engineering, East China Normal University

²KLATASDS-MOE in School of Statistics, East China Normal University

³NetEase Fuxi AI Lab, ⁴National University of Singapore

{jiayiwu, nuochen}@stu.ecnu.edu.cn, zhurenyu@corp.netease.com

qiushisun@u.nus.edu, {xiangli, mgao}@dase.ecnu.edu.cn

Abstract

Over the past few years, we have witnessed remarkable advancements in Code Pre-trained Models (CodePTMs). These models achieved excellent representation capabilities by designing structure-based pre-training tasks for code. However, how to enhance the absorption of structural knowledge when fine-tuning CodePTMs still remains a significant challenge. To fill this gap, in this paper, we present Structure-aware Fine-tuning (SAT), a novel structure-enhanced and plug-and-play fine-tuning method for CodePTMs. We first propose a *structure loss* to quantify the difference between the information learned by CodePTMs and the knowledge extracted from code structure. Specifically, we use the attention scores extracted from Transformer layer as the learned structural information, and the shortest path length between leaves in abstract syntax trees as the structural knowledge. Subsequently, multi-task learning is introduced to improve the performance of fine-tuning. Experiments conducted on four pre-trained models and two generation tasks demonstrate the effectiveness of our proposed method as a plug-and-play solution. Furthermore, we observed that SAT can benefit CodePTMs more with limited training data.

Keywords: Pre-trained Language Models, Model Tuning, Code Structure

1. Introduction

Pre-trained language Models (PTMs) (Liu et al., 2019) with Transformer architecture (Vaswani et al., 2017) have greatly improved a wide range of natural language processing (NLP) tasks. The success of PTMs in NLP also fosters the development of Code Pre-trained Models (CodePTMs) for processing programming languages (Sun et al., 2024b). Since Feng et al. (2020) proposed CodeBERT, a model with multi-layer bidirectional Transformer architecture pre-trained on NL-PL pairs, pre-training, and fine-tuning PTMs has gradually become the de facto paradigm for various code-related tasks leading to stunning performance.

Existing CodePTMs can be mainly categorized into two types: structure-free and structure-based models. The former, exemplified by CodeBERT (Feng et al., 2020), regards code snippets as a sequence of tokens, thereby ignoring the inherent structure of code. The latter, such as GraphCodeBERT (Guo et al., 2021) and CodeT5 (Wang et al., 2021) both incorporate structure-based pre-training tasks to capture more structural knowledge. Previous research (Xu and Zhu, 2022; Chen et al., 2023b,a) has shown that structure-based CodePTMs that learn structural knowledge possess more substantial code representation capabilities. These structure-based CodePTMs mainly focus on adding structure-related tasks to learn general structural knowledge during pre-training. However, as demonstrated by Wan et al. (2022)

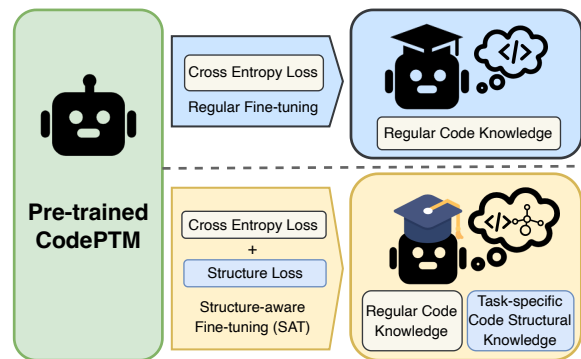


Figure 1: Compared to regular fine-tuning, CodePTMs can effectively capture task-specific code structure knowledge through structure-aware fine-tuning by structure loss.

and Chen et al. (2022), the CodePTMs tend to shift from general to task-specific structural knowledge in the fine-tuning stage. It further inspires us to think: *Can we enhance the absorption of task-specific structural knowledge in the fine-tuning stage for CodePTMs?*

Nevertheless, how to enhance the absorption of structural knowledge during the fine-tuning of CodePTMs still remains unaddressed (Sun et al., 2024a). Furthermore, most methods for learning structural code knowledge during the pre-training phase are not applicable during fine-tuning. For instance, if a fine-tuning approach were to imitate Guo et al. (2022) and Hu et al. (2020) by using traversed Abstract Syntax Trees (ASTs) as inputs,

* Corresponding author

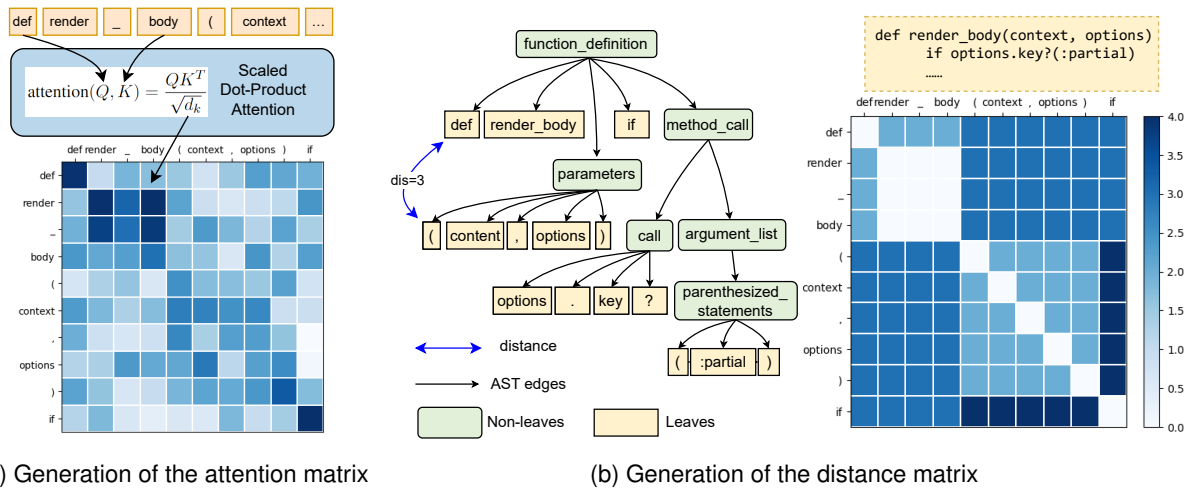


Figure 2: Schematic illustration of generation of the attention matrix and the distance matrix. (a) Generation of the attention matrix. The code sequence is input into the multi-head self-attention block of CodePTMs, then the scaled dot-product values are computed to generate attention matrix. (b) Generation of the distance matrix. We first parse the raw code into AST, and calculate the shortest path between leaves on the AST to generate distance matrix.

it would also necessitate constructing ASTs during inference. Other methods by Zügner et al. (2021) and Peng et al. (2021) incorporate structural knowledge by modifying the attention mechanism within Transformer blocks, involve heavy model modifications and cannot be seamlessly applied to existing pre-trained models. Lastly, Guo et al. (2021) and Wang et al. (2021) introduce structure-based pre-training tasks, which require predicting specific labels related to code structure, such as “data flow edges” or “token identifier.” However, these tasks are often unrelated to the fine-tuning tasks.

To address these issues, we present SAT, a structure-aware fine-tuning method for CodePTMs. As illustrated in Figure 1, SAT can help CodePTMs to capture structural knowledge during the fine-tuning phase effectively and seamlessly. Firstly, we parse the code into an abstract syntax tree (AST) and construct a distance matrix by calculating the shortest path between two leaves in AST. Then, we extract the attention matrix from the Transformer layer of CodePTMs. After that, inspired by Weng (2019) and Feydy et al. (2019), we introduce the Sinkhorn divergence to compute the difference between the two matrices as a new objective function, namely *structure loss*. Finally, during the fine-tuning phase, we optimize the model by jointly optimizing the structure loss and the downstream task objective function using a multi-task approach. We evaluate SAT on code summarization and translation tasks with four pre-trained models. Our contributions can be summarized as follows:

- We propose SAT, a novel multi-task learning method that enhances the absorption of structural knowledge for CodePTMs during the fine-

tuning stage. To the best of our knowledge, this is the first structure-aware fine-tuning method for CodePTMs.

- SAT can be easily implemented with the majority of Transformer-based CodePTMs as a plug-and-play solution.
- Our experiments involve four pre-trained models and two generation tasks, demonstrating that our proposed method can further improve the fine-tuning performance of CodePTMs. Additionally, SAT shows greater improvement in low-resource scenarios.

2. SAT

In this section, we introduce SAT. The detailed architecture of the SAT is illustrated in Figure 3.

2.1. Code Basics

As shown in Figure 2b, AST is a tree-like model used to represent the structure of a program, abstracting the source code into nodes and connecting edges. In this paper, we use Tree-sitter¹ to generate ASTs.

2.2. Distance Matrix and Attention Matrix

Chen et al. (2022) proposed that the model’s performance improves with its ability to explore structural knowledge, leading to a greater alignment between CodePTMs’ token-level attention scores and AST nodes’ pair-wise distances. Inspired by this

¹github.com/tree-sitter

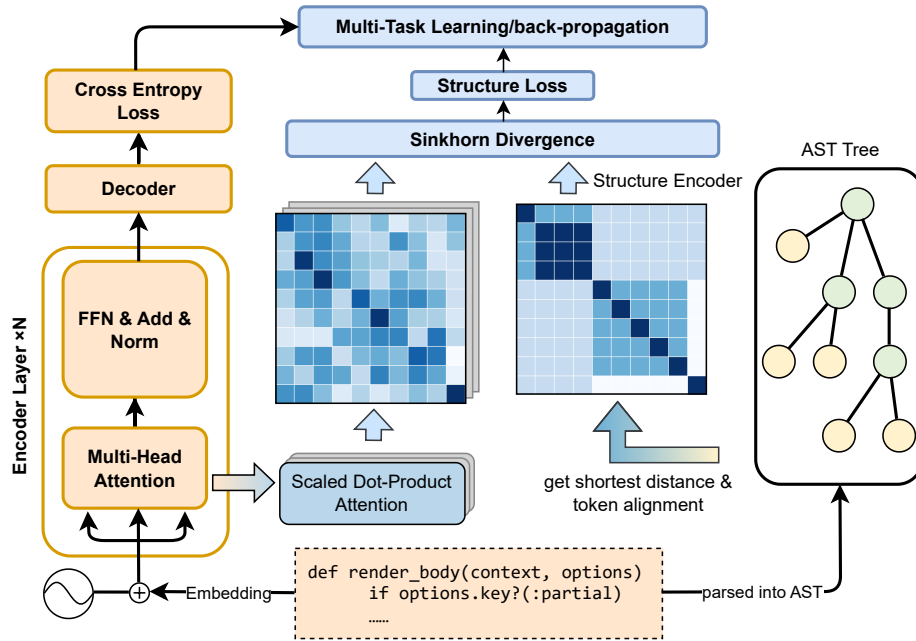


Figure 3: The architecture of the SAT method. Given a code snippet, we parsed it into an AST, and the distance matrix is obtained by computing the shortest distances between leaf nodes. The structural information is extracted using the struct encoder. The code snippet is then input into the CodePTMs, from which the attention matrix is extracted from the multi-head attention blocks. The sinkhorn divergence between the distance matrix and the attention matrix is calculated to obtain the structure loss.

work, we aim to quantify the disparity between the learned structural information by the pre-trained models and the structural knowledge from AST. To achieve this, we introduce the attention matrix and the distance matrix, respectively.

Figure 2 illustrates the process of generating the attention matrix and the distance matrix. The generation of the attention matrix is shown in Figure 2a, the raw code is input into the multi-head self-attention block of CodePTMs as a sequence of subtokens, where the scaled dot-product attention between subtokens is calculated. These attention values are extracted and used to construct the attention matrix. And Figure 2b illustrates the generation of the distance matrix. We parse the raw code into an AST, where each leaf node is derived from the raw code. To calculate the distances between tokens, we treat the AST as an undirected graph, where the distance between tokens corresponds to the length of the shortest path between the corresponding nodes in the AST. For example, the distance between token “def” and token “(” is 3. We extract the shortest distances between tokens from the AST to construct the distance matrix.

So far, the shapes of the attention matrix and the distance matrix are different. The attention matrix is at the subtoken level, while the distance matrix is at the token level. Subtokens are determined by the tokenizer, while tokens are determined by the tree-sitter. To facilitate the comparison between

the distance matrix and the attention matrix, we convert the distance matrix from the token level to the subtoken level, ensuring that both matrices have the same shape. Specifically, subtokens are generally contained within tokens, as shown in Figure 2b, subtoken “render”, “_” and “body” correspond to token “render_body”. By iterating through the sequences of subtokens and tokens, we get the mapping relationship between subtoken and token. The distance between subtokens is the distance between the corresponding tokens. In Figure 2b, for example, the subtoken “render” is contained within the token “render_body”, and the subtoken “:” is contained within the token “:partial”. The distance between them is 5.

2.3. Formal Definition of Structure Loss

Inspired by Weng (2019) and Feydy et al. (2019), we employ Wasserstein distance to quantify the differences between the attention matrix and distance matrix. The Wasserstein distance is a smooth metric for measuring the distance between two probability distributions, even when these distributions reside in non-overlapping lower dimensional manifolds (Weng, 2019). However, computing the Wasserstein distance entails high computational costs. To mitigate this, we utilize Sinkhorn divergences, which are positive and definite approximations of Wasserstein distances, and offer efficient batch computation on GPUs (Feydy et al., 2019).

In this paper, we leverage the Geomloss² for calculating Sinkhorn divergences.

Because the values of scaled dot-product attention and distance are not on the same scale, Sinkhorn divergences are not suitable for measuring the differences between two matrices that have values in different orders of magnitude. To address this issue, we introduce a linear layer that applies coefficients and biases to the distance matrix (Liu et al., 2013). This linear layer, referred to as the structure encoder, is treated as learnable parameters integrated into the model.

We treat the attention matrix and the distance matrix as two distributions. By applying the structure encoder to the distance matrix for scaling and biasing, we utilize Sinkhorn divergences to measure the differences between the two matrices to obtain the *structure loss*:

$$\mathcal{L}_i = \text{Sinkhorn divergences}(\mathbf{A}, \text{Linear}(\mathbf{D})), \quad (1)$$

$$\mathcal{L}_{\text{structure}} = \frac{\sum_{i=1}^h \mathcal{L}_i}{h} \quad (2)$$

where $\mathbf{A}, \mathbf{D} \in \mathbb{R}^{n \times n}$, \mathbf{A} denotes the attention matrix, \mathbf{D} represents the distance matrix, n represents the length of the input sequence in CodePTMs, and h represents the number of attention heads.

Among the Transformer layers in CodePTMs, the first layer captures the most structural information (Chen et al., 2022). Therefore, we extract the attention values from the first Transformer layer and utilize them as the attention matrix.

We draw inspiration from the concept of multi-task fine-tuning (Kalyan et al., 2021) to jointly optimize the objective function of downstream tasks and the structure loss. This approach enables the model to learn the structural information of the code during the backpropagation phase. The final objective function is followed as:

$$\mathcal{L}_{\text{final}} = \mathcal{L}_{\text{task}} + \alpha \cdot \mathcal{L}_{\text{structure}} \quad (3)$$

where $\mathcal{L}_{\text{task}}$ represents the objective function of the downstream task, such as cross entropy loss for code translation and code summarization. And α is a hyperparameter that controls the weight of the structure loss.

As a plug-and-play method, the *structure loss* can be easily applied when needed and is applicable to majority Transformer-based CodePTMs, demonstrating its generality.

3. Experiments

3.1. Tasks and Metrics

To evaluate the ability of our proposed method SAT, we select two challenging tasks in the CodeXGLUE

(Lu et al., 2021) Benchmark: code summarization and code translation, to assess cross-modal generation and code-to-code generation capabilities, respectively.

Code summarization aims to generate a natural language description for a given programming language snippet. We evaluated the effectiveness of our method using the CodeSearchNet (Husain et al., 2019) dataset for code summarization across five programming languages: Ruby, JavaScript, Go, Python, and Java. We evaluate the results using the smoothed BLEU-4 (Lin and Och, 2004) metric.

Code translation involves translating a code from one programming language to a different one. We conducted experiments on this task using the dataset provided by CodeXGLUE, which consists of ten thousand pairs of Java and C# code snippets with equivalent functionality. We follow the setting of CodeXGLUE and employ BLEU-4 and exact match (EM) as evaluation metrics.

3.2. Backbone models

We implement SAT to four representative PTMs, encompassing two distinct types: encoder-only and encoder-decoder models.

For the encoder-only architecture, we employed RoBERTa (Liu et al., 2019), a pre-trained model trained on a large-scale text corpus using masked language modeling (MLM). Additionally, we choose two RoBERTa-based CodePTMs, CodeBERT (Feng et al., 2020) and GraphCodeBERT (Guo et al., 2021).

In the case of the encoder-decoder architecture, we employ CodeT5 (Wang et al., 2021), a state-of-the-art CodePTM based on the T5 framework (Rafael et al., 2020), known for its exceptional performance on various tasks within the CodeXGLUE benchmark (Lu et al., 2021).

3.3. SAT Effectiveness

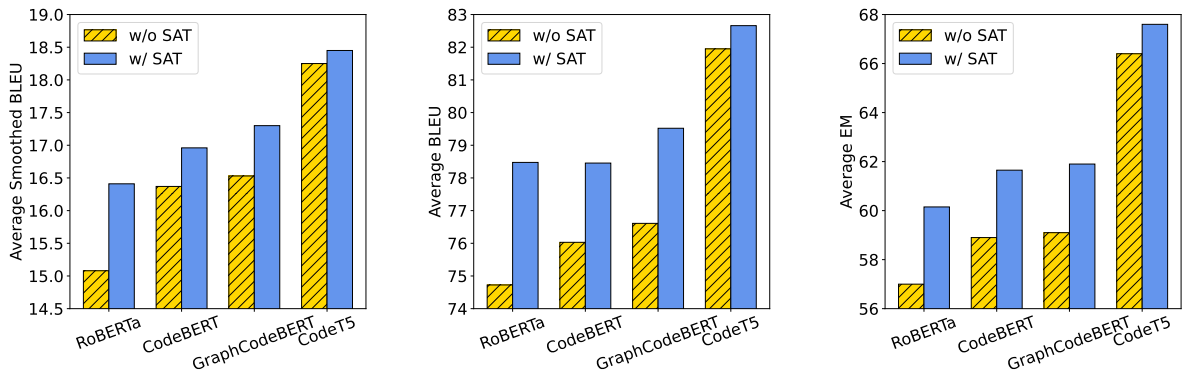
The results of the SAT method on the code summarization and code translation tasks are presented in Table 1. Upon analyzing these experimental results, we discover that:

- **SAT can enhance the absorption of structural knowledge for CodePTMs in the fine-tuning stage:** Our proposed SAT approach obtains consistent improvements across the RoBERTa, CodeBERT, GraphCodeBERT, and CodeT5 models. In some scenarios, weaker models enhanced by our method can outperform stronger models, e.g., on code summarization task, RoBERTa enhanced by SAT even surpasses the unaugmented CodeBERT. Similarly, CodeBERT’s performance

²<https://www.kernel-operations.io/geomloss/index.html>

Model	Code summarization (Smoothed BLEU-4)						Java to C#		C# to Java	
	Ruby	JavaScript	Go	Python	Java	Overall	BLEU	EM	BLEU	EM
RoBERTa	11.17	11.90	17.72	18.14	16.47	15.08	77.46	56.10	71.99	57.90
RoBERTa+SAT	11.82 (+0.65)	15.68 (+3.78)	17.85 (+0.13)	18.49 (+0.35)	18.21 (+1.74)	16.41 (+1.33)	80.28 (+2.82)	59.30 (+3.20)	76.67 (+4.68)	61.00 (+3.10)
CodeBERT	12.16	14.90	18.07	19.06	17.65	16.37	79.92	59.00	72.14	58.80
CodeBERT+SAT	12.50 (+0.34)	16.34 (+1.44)	18.32 (+0.25)	19.02 (-0.04)	18.64 (+0.99)	16.96 (+0.59)	80.51 (+0.59)	61.30 (+2.30)	76.40 (+4.26)	62.00 (+3.20)
GraphCodeBERT	12.39	14.81	18.41	18.06	19.00	16.53	80.58	59.40	72.64	58.80
GraphCodeBERT+SAT	13.16 (+0.77)	16.33 (+1.52)	18.63 (+0.22)	19.24 (+1.18)	19.15 (+0.15)	17.30 (+0.77)	81.74 (+1.16)	62.30 (+2.90)	77.30 (+4.66)	61.50 (+2.70)
CodeT5	15.24	16.16	19.56	20.01	20.31	18.25	84.03	65.90	79.87	66.90
CodeT5+SAT	15.51 (+0.27)	16.20 (+0.04)	19.73 (+0.17)	20.34 (+0.33)	20.48 (+0.17)	18.45 (+0.20)	84.99 (+0.96)	67.50 (+1.60)	80.33 (+0.46)	67.70 (+0.80)

Table 1: The results of SAT on code summarization and code translation tasks. The left half shows the smoothed BLEU-4 scores on the code summarization tasks, while the right half shows the BLEU-4 scores and exact match (EM) accuracy for code translation tasks. “+SAT” indicates that the model was enhanced with the SAT method. Better results in the same task are bolded.



(a) The average smoothed BLEU-4 scores on the code summarization (b) The average BLEU-4 scores for the code translation (c) The average exact match (EM) for the code translation.

Figure 4: Average results of SAT on code summarization and code translation tasks.

after SAT augmentation outperforms unaugmented GraphCodeBERT. Notably, significant performance gains are observed in some models and tasks. For instance, RoBERTa+SAT achieves a smooth BLEU-4 score improvement of 3.78 on the JavaScript dataset for code summarization. And for the task of translating C# to Java, RoBERTa+SAT achieves an exact match improvement of 4.68.

- **SAT can improve more with weaker models:** Figure 4 illustrates the performance improvement of the SAT method across different backbone models. Among these models, RoBERTa exhibits the most substantial enhancement, followed by CodeBERT and GraphCodeBERT, while CodeT5 shows the least improvement. This is likely due to the fact that RoBERTa is trained on pure textual corpora and lacks explicit code structural knowledge during the pre-training phase. Consequently, incorporating SAT during fine-tuning

significantly enhances the model’s performance. In contrast, CodeT5 already acquires extensive structural code information during pre-training through identifiers modeling, resulting in a smaller impact from SAT. These findings demonstrate that SAT effectively assists PTMs in learning structural code knowledge during fine-tuning, with models possessing less pre-trained code knowledge experiencing more notable performance gains.

3.4. Exploration of Different Training Data Scales

In order to evaluate the ability of the SAT method to extract code structural information from limited data, we conducted experiments using both the CodeBERT and CodeT5 models on code summarization and code translation tasks with different training data scale. We sampled 20%, 40%, 60%, and 80% of the training data, and the results are presented in Table 2. From the results, we observe

Sample rate	Model	Code summarization		Code Translation		Model	Code summarization		Code Translation	
		Ruby	Python	BLEU	EM		Ruby	Python	BLEU	EM
20%	CodeBERT	11.64	18.66	75.66	56.10	CodeT5	15.18	19.90	81.21	61.10
	CodeBERT+SAT	11.98	18.98	76.58	57.20	CodeT5+SAT	15.89	20.27	81.97	61.90
		(+0.34)	(+0.32)	(+0.92)	(+1.10)		(+0.71)	(+0.37)	(+0.76)	(+0.80)
40%	CodeBERT	12.09	18.98	78.76	59.00	CodeT5	15.24	19.96	84.13	65.80
	CodeBERT+SAT	12.27	19.00	79.55	60.40	CodeT5+SAT	15.66	20.17	84.52	66.00
		(+0.18)	(+0.02)	(+0.79)	(+1.40)		(+0.42)	(+0.21)	(+0.39)	(+0.2)
60%	CodeBERT	12.16	18.87	79.39	59.40	CodeT5	15.19	20.18	84.41	66.20
	CodeBERT+SAT	12.37	19.03	80.03	59.50	CodeT5+SAT	15.35	20.24	84.58	66.80
		(+0.21)	(+0.16)	(+0.64)	(+0.10)		(+0.16)	(+0.06)	(+0.17)	(+0.60)
80%	CodeBERT	11.87	18.99	80.32	61.50	CodeT5	15.36	20.20	84.68	66.40
	CodeBERT+SAT	11.75	19.11	80.88	61.90	CodeT5+SAT	15.56	20.21	84.77	66.20
		(-0.12)	(+0.12)	(+0.56)	(+0.40)		(+0.20)	(+0.01)	(+0.09)	(-0.20)

Table 2: Result of fine-tuning on a limited training dataset. The highest performance improvement is bolded. (Code Translation refers to the task of translating Java to C#.)

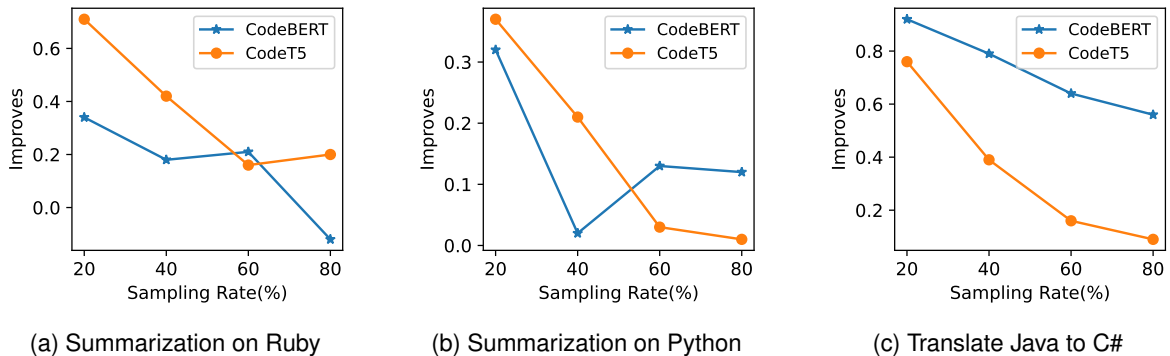


Figure 5: Performance improvement of fine-tuning with SAT on limited training datasets.

that the SAT method consistently enhances the performance of the model across various training data scales.

Significantly, as shown in Figure 5, we observe that **the SAT method yields more substantial performance improvements as the training data scale decreases**. We believe this is because, under the influence of the SAT method, the language model can learn a sufficient amount of structural information without requiring a large amount of training data. This ability becomes evident when the language model is fine-tuned with a small-scale dataset, as it can learn a considerable amount of code structural information even in the absence of sufficient semantic knowledge, thus compensating for the limitations in semantic understanding. This indicates that SAT can better assist models in enhancing performance in low-resource scenarios.

3.5. Did the SAT really learn the structural information?

Analysis from the perspective of structure loss
We optimize the structure loss by multi-task fine-tuning. The curve of structure loss with the number of training iterations is shown in Figure 6. We observe that the structure loss gradually decreases as the number of iterations increases. This indi-

cates a diminishing difference between the attention and distance matrices, suggesting that the model is learning structural information. Notably, in the early stages of fine-tuning, the structure loss of CodeT5+SAT is consistently lower than that of GraphCodeBERT+SAT. This observation indicates that CodeT5 has learned more structural knowledge during the pre-training phase.

Analyzing from CAT-probing CAT-probing (Chen et al., 2022) is a probing method to quantitatively interpret how CodePTMs attend code structure. It defines a new metric CAT-score to measure the commonality between the token-level attention scores generated in CodePTMs and the pair-wise distances between corresponding AST nodes. The higher the CAT-score, the stronger the ability of CodePTMs to capture code structure.

To assess whether SAT enhances the absorption of code structural knowledge, we selected the RoBERTa, CodeBERT, and GraphCodeBERT models and conducted code summarization tasks on Ruby and Java datasets and computed the CAT-score for each model with and without the incorporation of SAT.

As shown in Figure 7, we observed that all models exhibited an increase in their CAT-scores upon the introduction of SAT. This indicates that SAT has

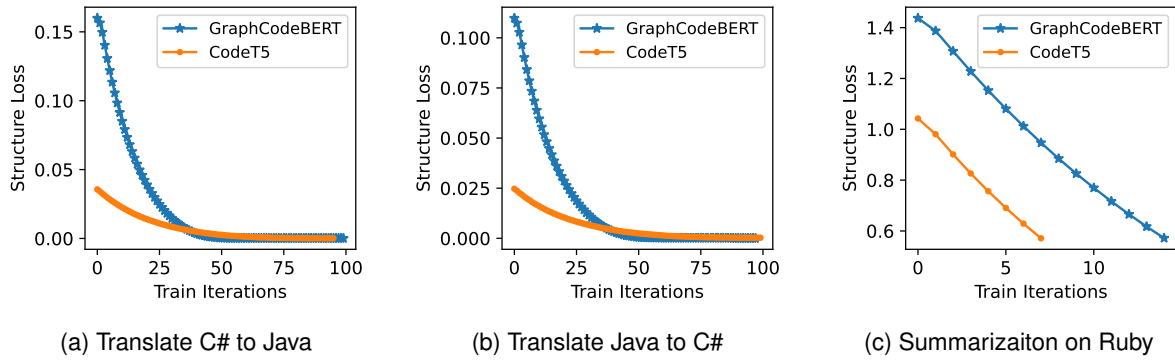


Figure 6: Curve of structure loss

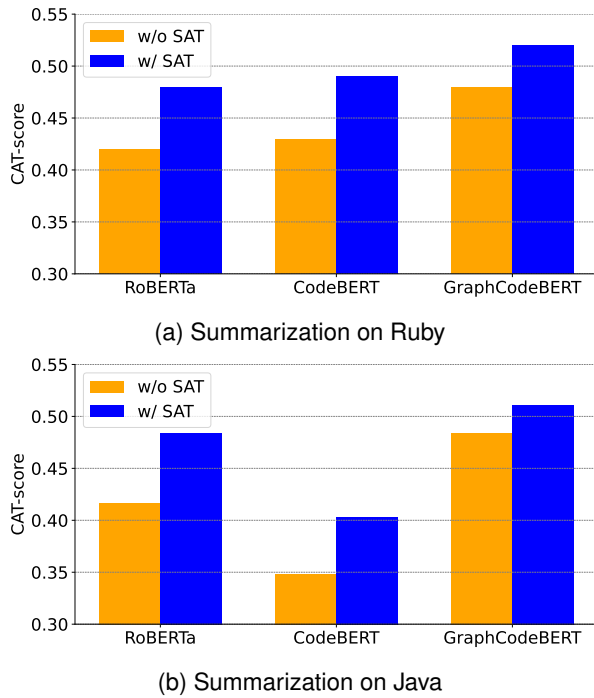


Figure 7: Comparative of CAT Scores with and without the SAT

the ability to enhance pre-trained models to capture a greater amount of code structural knowledge. The degree of improvement in CAT-scores is correlated with the level of code knowledge acquired during the pre-training phase by the backbone models, with RoBERTa exhibiting the greatest improvement, followed by CodeBERT and GraphCodeBERT. This indicates that SAT is particularly effective in assisting models with relatively weaker performance, thus compensating for their deficiency in acquiring an adequate level of code structural knowledge during the training phase. This observation aligns with the findings presented in Section 3.3.

Analyzing from a case study We select two representative cases to illustrate the difference in model outputs with and without the SAT method.

As shown in Table 3, we provide two examples of the test data of translating C# to Java. In the first example, the output structure with the SAT method is consistent with the correct answer, except for the token “value”, which differs from the correct answer, while the output structure without the SAT method is entirely dissimilar to the correct answer. In the second example, the output with the SAT method is identical to the correct answer, whereas the output without the SAT method has misaligned variable assignments, indicating a lack of learned structural information from the input code sequence. These two examples demonstrate that SAT aids CodePTMs in acquiring more structural knowledge.

4. Related Work

4.1. Pre-Trained Language Models for Programming Languages

With the great success of pre-trained language models in natural language processing, transformer-based CodePTMs (Guo et al., 2021; Wang et al., 2021, 2023) have significantly propelled the development of code intelligence. Existing CodePTMs can be mainly categorized into two classes: struct-free and struct-based. In the struct-free category, Feng et al. (2020) first proposes CodeBERT, a bimodal pre-trained model based on RoBERTa, which leverages masked language modeling and replaced token detection to learn NL-PL bimodal information. Ahmad et al. (2021) follows the BART architecture and is pre-trained with denoising autoencoding on Python/Java code and NL corpus. In the struct-aware category, GraphCodeBERT (Guo et al., 2021) builds upon CodeBERT and incorporates the inherent structure of code by utilizing data flow to enhance code representation. CodeT5 (Wang et al., 2021) builds on a unified encoder-decoder architecture based on T5 (Raffel et al., 2020) and supports various downstream tasks and multitask learning. It introduces a novel

	Case 1	Case 2
Source	<pre>public override V next() { return this.nextEntry().value; }</pre>	<pre>public SinglePositionTokenStream(string word) { termAtt = addAttribute(CharTermAttribute.class); posIncrAtt = addAttribute(PositionIncrementAttribute.class); this.word = word; }</pre>
Target	<pre>public V next() { return super.nextEntry().getValue(); }</pre>	<pre>public SinglePositionTokenStream(String word) { termAtt = addAttribute(CharTermAttribute.class); posIncrAtt = addAttribute(PositionIncrementAttribute.class); this.word = word; returned = true; }</pre>
RoBERTa	<pre>public V next() { V v = nextValue(); advance(); return v; }</pre>	<pre>public SinglePositionTokenStream(String word) { termAtt = addAttribute(CharTermAttribute.class); posIncrAtt = word; returned = true; }</pre>
RoBERTa +SAT	<pre>public V next() { return super.nextEntry().value; }</pre>	<pre>public SinglePositionTokenStream(String word) { termAtt = addAttribute(CharTermAttribute.class); posIncrAtt = addAttribute(PositionIncrementAttribute.class); this.word = word; returned = true; }</pre>

Table 3: Case study

identifier-aware pre-training task that considers the identifiers from code. UniXcoder (Guo et al., 2022) adapts UniLM (Dong et al., 2019) and enhances code representation by employing AST and code comments, using mask attention matrices with prefix adapters to control the model’s behavior. Xu and Zhu (2022) has found that CodePTMs with learned structural knowledge have more powerful code representation capabilities. We pursue this research line, adding structural knowledge in the fine-tuning phase.

4.2. Neural Networks With Code Structure

In addition to the CodePTMs discussed in Section 4.1 that attempt to incorporate structural-related pre-training tasks during the pre-training process, other studies have explored different approaches for introducing code structural knowledge. Hu et al. (2023) proposed DeepM to measure code maintainability by exploiting the lexical semantics and the structural features of text in source code. UnixCoder(Guo et al., 2022) and Hybrid-DeepCom(Hu et al., 2020) take traversed ASTs as inputs during the pre-training phase. However, this approach is not applicable for fine-tuning, as the inputs for fine-tuning and inference should ideally align. Great (Hellendoorn et al., 2020), Code Transformer (Zügner et al., 2021), and TPTrans (Peng et al., 2021) incorporate structural knowledge into the attention computation, enabling the model to integrate program context and structure information. Great utilizes edge type embedding in the program graph, Code Transformer leverages node distances across the AST, and TPTrans employs relative and absolute path encoding. However, these Transformer-based methods require heavy model modifications. As a result, they cannot be readily transferred to existing pre-trained

models and are not suitable for implementation during the fine-tuning phase. To address this issue, we incorporate structural information through the structure loss and multi-task fine-tuning, enabling a plug-and-play approach for our method.

4.3. Fine-tuning Approaches for Pre-trained Models

Existing methods for fine-tuning pre-trained models can be categorized into five types: vanilla fine-tuning, intermediate fine-tuning, multi-task fine-tuning, parameter-efficient fine-tuning, and prompt-based fine-tuning (Kalyan et al., 2021). For programming language processing, some researchers (Bogomolov et al., 2022; Wang et al., 2022; Ayupov and Chirkova, 2022) have explored the fine-tuning of CodePTMs based on the methods above. Bogomolov et al. (2022) conducted vanilla fine-tuning under different settings and observed that performance improves as the training data becomes more relevant to the task domain. Wang et al. (2022) conducted prompt-based fine-tuning on CodePTMs and observed better results than vanilla fine-tuning. Ayupov and Chirkova (2022) addressed the issue of deploying large-scale pre-trained models in integrated development environments by employing parameter-efficient fine-tuning. Sun et al. (2023) endeavored to capture the inherent connections between different programming languages and code-related tasks to enhance the efficacy of fine-tuning. Recently, Liu et al. (2024) proposed an efficient yet effective paradigm to adapt frozen large-scale PTMs to specific downstream tasks by learns dense representations. To the best of our knowledge, there has yet to be any prior research on applying multi-task fine-tuning specifically for code pretraining models. In this study, we fine-tune CodePTMs using the multi-task fine-tuning approach.

5. Conclusion

In this paper, we propose SAT, a novel multi-task fine-tuning approach that enhances the structural knowledge of CodePTMs during the fine-tuning phase. We quantify the difference between the attention matrix from the Transformer block and the distance matrix extracted from the AST as the structure loss using Sinkhorn divergences. By employing multi-task learning, CodePTMs can learn code structural knowledge during the fine-tuning phase. This approach does not require modifying the model architecture, enabling SAT to possess strong portability. To validate the effectiveness of SAT, We conducted experiments on four pre-trained models and two code generation tasks. Experimental results demonstrate that our approach enhances the representational capacity of CodePTMs and benefits tasks in low-resource scenarios.

6. Limitations

SAT focuses on code-to-code and code-to-text tasks but cannot be applied to text-to-code tasks. For encoder-only models, we can only leverage the structural knowledge absorption of the source sequence during fine-tuning, while we cannot absorb structural information from the target sequence due to the absence of a corresponding attention matrix. In our future work, we will explore structural enhancement methods suitable for text-to-code tasks.

7. Ethical Considerations

The proposed method does not exhibit potential risks. All the scientific artifacts used or created in this study are duly cited and properly licensed, and their usage aligns with their intended purpose.

8. Acknowledgments

This work has been supported by the National Natural Science Foundation of China under Grant No. 62377012.

9. Bibliographical References

- Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. [Unified pre-training for program understanding and generation](#). In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 2655–2668. Association for Computational Linguistics.
- Shamil Ayupov and Nadezhda Chirkova. 2022. [Parameter-efficient finetuning of transformers for source code](#). *CoRR*, abs/2212.05901.
- Egor Bogomolov, Sergey Zhuravlev, Egor Spirin, and Timofey Bryksin. 2022. [Assessing project-level fine-tuning of ML4SE models](#). *CoRR*, abs/2206.03333.
- Nuo Chen, Qiushi Sun, Jianing Wang, Ming Gao, Xiaoli Li, and Xiang Li. 2023a. [Evaluating and enhancing the robustness of code pre-trained models through structure-aware adversarial samples generation](#). In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14857–14873, Singapore. Association for Computational Linguistics.
- Nuo Chen, Qiushi Sun, Jianing Wang, Xiang Li, and Ming Gao. 2023b. [Pass-tuning: Towards structure-aware parameter-efficient tuning for code representation learning](#). In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 577–591, Singapore. Association for Computational Linguistics.
- Nuo Chen, Qiushi Sun, Renyu Zhu, Xiang Li, Xuesong Lu, and Ming Gao. 2022. [Cat-probing: A metric-based approach to interpret how pre-trained models for programming language attend code structure](#). In *Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 4000–4008. Association for Computational Linguistics.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. [BERT: pre-training of deep bidirectional transformers for language understanding](#). In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.
- Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. [Unified language model pre-training for natural language understanding and generation](#). In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 13042–13054.

- 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](#). In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 1536–1547. Association for Computational Linguistics.
- Jean Feydy, Thibault Sjourn, Franois-Xavier Vialard, Shun-ichi Amari, Alain Trouv, and Gabriel Peyr. 2019. [Interpolating between optimal transport and MMD using sinkhorn divergences](#). In *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*, volume 89 of *Proceedings of Machine Learning Research*, pages 2681–2690. PMLR.
- Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. [Unixcoder: Unified cross-modal pre-training for code representation](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 7212–7225. Association for Computational Linguistics.
- Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin B. Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. [Graphcodebert: Pre-training code representations with data flow](#). In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.
- Vincent J. Hellendoorn, Charles Sutton, Rishabh Singh, Petros Maniatis, and David Bieber. 2020. [Global relational models of source code](#). In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.
- Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2020. [Deep code comment generation with hybrid lexical and syntactical information](#). *Empir. Softw. Eng.*, 25(3):2179–2217.
- Yamin Hu, Hao Jiang, and Zongyao Hu. 2023. [Measuring code maintainability with deep neural networks](#). *Frontiers Comput. Sci.*, 17(6):176214.
- Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. [Codesearchnet challenge: Evaluating the state of semantic code search](#). *CoRR*, abs/1909.09436.
- Katikapalli Subramanyam Kalyan, Ajit Rajasekharan, and Sivanesan Sangeetha. 2021. [AMMUS : A survey of transformer-based pretrained models in natural language processing](#). *CoRR*, abs/2108.05542.
- Anjan Karmakar and Romain Robbes. 2021. [What do pre-trained code models know about code?](#) In *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*, pages 1332–1336. IEEE.
- Chin-Yew Lin and Franz Josef Och. 2004. [ORANGE: a method for evaluating automatic evaluation metrics for machine translation](#). In *COLING 2004, 20th International Conference on Computational Linguistics, Proceedings of the Conference, 23-27 August 2004, Geneva, Switzerland*.
- Hui Lin, Xiaopeng Hong, Zhiheng Ma, Xing Wei, Yunfeng Qiu, Yaowei Wang, and Yihong Gong. 2021. [Direct measure matching for crowd counting](#). In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 837–844. ijcai.org.
- Guangcan Liu, Zhouchen Lin, Shuicheng Yan, Ju Sun, Yong Yu, and Yi Ma. 2013. [Robust recovery of subspace structures by low-rank representation](#). *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(1):171–184.
- 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 pre-training approach](#). *CoRR*, abs/1907.11692.
- Yitao Liu, Chenxin An, and Xipeng Qiu. 2024. [Y-tuning: an efficient tuning paradigm for large-scale pre-trained models via label representation learning](#). *Frontiers Comput. Sci.*, 18(4).
- Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. 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](#). In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.
- Han Peng, Ge Li, Wenhan Wang, Yunfei Zhao, and Zhi Jin. 2021. [Integrating tree path in transformer](#)

- for code representation. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 9343–9354.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. [Exploring the limits of transfer learning with a unified text-to-text transformer](#). *J. Mach. Learn. Res.*, 21:140:1–140:67.
- Qiushi Sun, Nuo Chen, Jianing Wang, Xiang Li, and Ming Gao. 2023. [Transcoder: Towards unified transferable code representation learning inspired by human skills](#).
- Qiushi Sun, Nuo Chen, Jianing Wang, and Xiaoli Li. 2024a. [Rethinking the role of structural information: How it enhances code representation learning?](#) In *International Joint Conference on Neural Networks, IJCNN 2024, Yokohama, Japan, June 30 - July 5, 2024*. IEEE.
- Qiushi Sun, Zhirui Chen, Fangzhi Xu, Kanzhi Cheng, Chang Ma, Zhangyue Yin, Jianing Wang, Chengcheng Han, Renyu Zhu, Shuai Yuan, Qipeng Guo, Xipeng Qiu, Pengcheng Yin, Xiaoli Li, Fei Yuan, Lingpeng Kong, Xiang Li, and Zhiyong Wu. 2024b. [A survey of neural code intelligence: Paradigms, advances and beyond](#).
- Sergey Troshin and Nadezhda Chirkova. 2022. [Probing pretrained models of source codes](#). In *Proceedings of the Fifth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2022, Abu Dhabi, United Arab Emirates (Hybrid), December 8, 2022*, pages 371–383. Association for Computational Linguistics.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. [Attention is all you need](#). *Advances in neural information processing systems*, 30.
- Yao Wan, Wei Zhao, Hongyu Zhang, Yulei Sui, Guandong Xu, and Hai Jin. 2022. [What do they capture? - A structural analysis of pre-trained language models for source code](#). In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*, pages 2377–2388. ACM.
- Chaozheng Wang, Yuanhang Yang, Cuiyun Gao, Yun Peng, Hongyu Zhang, and Michael R. Lyu. 2022. [No more fine-tuning? an experimental evaluation of prompt tuning in code intelligence](#). In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*, pages 382–394. ACM.
- Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi D. Q. Bui, Junnan Li, and Steven C. H. Hoi. 2023. [Codet5+: Open code large language models for code understanding and generation](#). *CoRR*, abs/2305.07922.
- Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. 2021. [Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation](#). In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 8696–8708. Association for Computational Linguistics.
- Lilian Weng. 2019. [From GAN to WGAN](#). *CoRR*, abs/1904.08994.
- Yichen Xu and Yanqiao Zhu. 2022. [A survey on pretrained language models for neural code intelligence](#). *CoRR*, abs/2212.10079.
- Renyu Zhu, Lei Yuan, Xiang Li, Ming Gao, and Wenyan Cai. 2022. [A neural network architecture for program understanding inspired by human behaviors](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5142–5153, Dublin, Ireland. Association for Computational Linguistics.
- Daniel Zügner, Tobias Kirschstein, Michele Catasta, Jure Leskovec, and Stephan Günnemann. 2021. [Language-agnostic representation learning of source code from structure and context](#). In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.