Towards Accurate Knowledge Transfer between Transformer-based Models for Code Summarization

Chaochen Shi [1], Yong Xiang [2], Jiangshan Yu [3], and Longxiang Gao [4]
[1, 2] School of Information Technology, Deakin University, Australia
[3] Faculty of Information Technology, Monash University, Australia
[4] Qilu University of Technology (Shandong Academy of Sciences), China
Email: {shicha, yong.xiang}@deakin.edu.au, j.yu.research@gmail.com, longx.gao@gmail.com

***Abstract*—Automatic code summarization generates high-level natural language descriptions of code snippets, which can benefit software maintenance and code comprehension. Recently, Transformer-based models achieved state-of-the-art performance on code summarization tasks. However, there are data gaps in neural model training for some programming languages. To fill this gap, we propose a novel transfer learning approach to accurately transfer knowledge between Transformer-based models. We train a discriminator to identify which heads of the multi-head attention module should be transferred. On this basis, we define a transfer strategy of parameter matrices. We evaluated the proposed transfer learning approach on four state-of-the-art Transformer-based code summarization models. Experimental results show that models with transferred knowledge outperform original models up to 10.70% in BLEU, 5.36% in ROUGE-L, and 4.34% in METEOR.**

Keywords-Transfer Learning; Code Summarization

I. INTRODUCTION

Automatic code summarization is a seq2seq (sequence to sequence) task of automatically generating natural sentences to describe a code snippet. Due to the general lack of high-quality source code comments in software development, automatic code summarization tools are of great significance in helping developers understand code and improve development efficiency. In recent years, due to the outstanding performance of Transformer [7] on seq2seq tasks, many studies on automatic code summarization leverage Transformer-based architectures [1, 2, 5, 8]. These state-of-the-art studies have proven that Transformer architecture is highly effective in capturing dependencies between code tokens. However, such neural code summarization models require a large number of <*code, comment*> pairs as ground truth data for training. For mainstream programming languages such as Java and Python, there are already large public datasets such as SIT [3] and CodeSearchNet [4] with millions of records. However, there is a lack of available large corpus for programming languages with smaller communities, such as Solidity that specializes in smart contract development on blockchain platforms. It leads to the poor performance of directly training neural models on such programming languages.

A potential solution to the gaps mentioned above is to transfer specific knowledge from well-trained code summarization models to less-trained models through transfer learning. The idea comes from the intuition that shared features (types, syntax, object-oriented characteristics, etc.) exist in different programming languages. As for Transformer, each parallel attention layer (head) pays attention to an individual subspace, which means that some heads may focus on shared features among different programming languages. This paper proposes a novel transfer learning approach to accurately transfer the Transformer's multi-head attention between programming languages. We train a discriminator *D* to filter heads that focus on similar features in transform-based models of different programming languages. Then we transfer the corresponding model parameters from the source domain to the target domain according to the similarity weights of heads. Next, we train the model of the target domain to fit its dataset. Our contributions are as follows:

- We propose the transfer learning approach dedicated to conducting accurate knowledge transfer between Transformer-based code summarization models, which can improve the performance of models in the target domain and speed up the training process.
- Taking Python and Solidity as examples, we compare the performance with and without the proposed transfer learning approach on four state-of-the-art Transformer-based models. The experimental results demonstrate the maximum improvement reaches 10.70%, 5.36%, and 4.34% in BLEU, ROUGE-L, and METEOR, respectively.

II. BACKGROUND AND RELATED WORK

Attention mechanism has become an integral part of sequence modeling, allowing the model to select the more critical information to the current task. The self-attention mechanism is a variant of the attention mechanism, which reduces the dependence on external information and better captures the internal correlations of data or features. The attention is calculated as equation 1.

$$Attn(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

Where *Q, K, V* are three tensors of the input, d_k is the dimension of *Q* and *K*. In self-attention mechanism, *Q*=*K*=*V*.

DOI reference number: 10.18293/SEKE2022-111

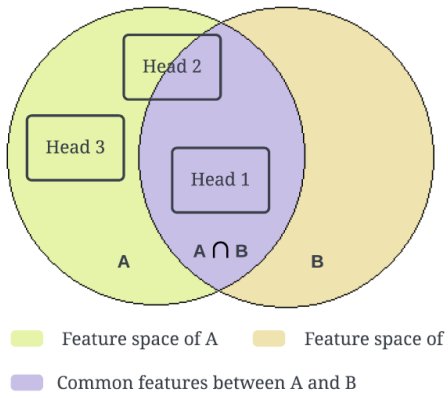

Figure 1. The feature space covered by different heads. Apparently, head 1 covers more shared features between A and B than head 2, 3.

Transformer is an encoder-decoder model relying on attention mechanism as the main component, which enables parallel computing and improves the feature extraction ability. As the state-of-the-art solutions to code summarization tasks, Transformer-based models have been widely studied. Ahmad [1] used Transformers to extract textual and structural features from code token sequence and AST (Abstract Syntax Tree), respectively. Clement [2] proposed Pymt5, a Python method text-to-text transfer Transformer, which is trained to translate between all pairs of Python method feature combinations. Liu [5] proposed a joint summary generation model based on improving Transformer, adding pointer mechanism and consistency loss function to keep the original meaning in generated sentences as much as possible. Wang [8] designed a structure encoding algorithm to represent hierarchical code structures. They combined it with BERT (a Transformer-based pre-trained model) to better extract code structural features. These mentioned works show Transformer-based models outperform existing RNN- and LSTM-based models by a large margin, playing the leading role in code summarization tasks recently.

III. Our Approach

The knowledge mentioned in this paper is the ability of the model to extract features from the input. Supposing there are two programming languages *A* and *B*, where A has a well-trained transformer-based code summarization model M_A while *B* has a structurally identical but undertrained model M_B. Our target is accurately transferring knowledge from M_A to M_B to facilitate the training process of M_B. Our research questions include:

- RQ1: How to identify the knowledge (heads) we need to transfer?
- RQ2: How to transfer the selected knowledge between Transformer-based models?

We address RQ1 and RQ2 in section III-A and III-B, respectively.

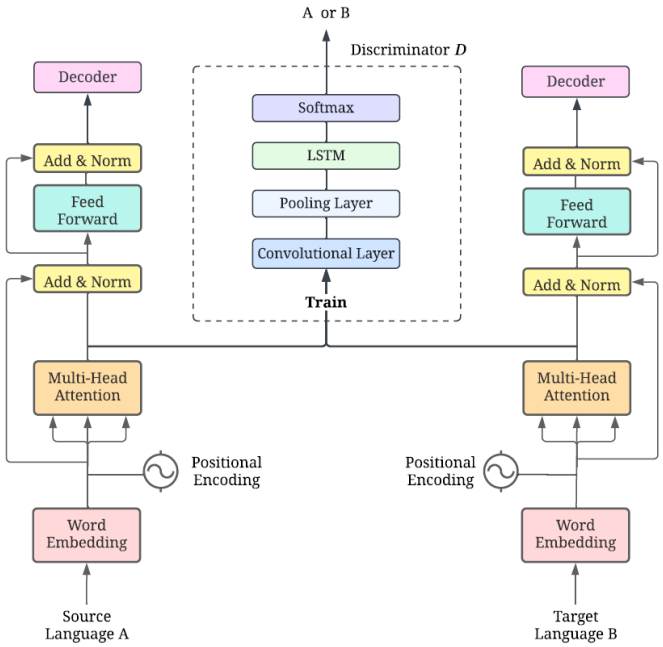

Figure 2. The structure and training process of the discriminator D. The left and right half are of the same Transformer M_A trained by source language A. D receives data from the multi-head attention module and tries to identify whether inputs are from language *A* or *B*.

A. Transferable Knowledge Identification

To address RQ1, we need a strategy to identify transferable knowledge which fits both source domain and target domain. For Transformer-based architectures with *h* heads, the model learns vector representations $z_1, z_2, \dots, z_h$ from each head. Then the multi-head attention representation Z is calculated as equation 2.

$$MultiAttnZ = Concat(z_1, z_2, \dots, z_h)W^O \quad (2)$$

where

$$z_i = Attn\left(QW_i^Q, KW_i^K, VW_i^V\right) \quad (3)$$

Where W_i^Q, W_i^K, W_i^V and W^O are parameter matrices of linear layers as Fig. 3 shows. The multi-head attention module of Transformer allows the model to jointly attend to information from *h* different representation subspaces at different positions [7]. As Fig.1 shows, some heads may focus on more shared features between *A* and *B* than others, which means such heads can contribute more transferable knowledge. We call them transferable heads.

Supposing we already have a well-trained Transformer-based model M_A for *A*. We train a discriminator *D* to identify transferable heads in M_A. As Fig. 2 shows, *D* can be regarded as a binary classification model which receives a code representation *Z* of M_A 's multi-head attention module and tries to identify whether *Z* is from *A* or *B*. Since the code token sequence is generally long, we use a Convolutional layer and a Pooling layer (average pooling) to reduce the dimensionality of the input. Then the input goes through an LSTM layer and a Softmax layer, outputs the probability *P* of being classified into

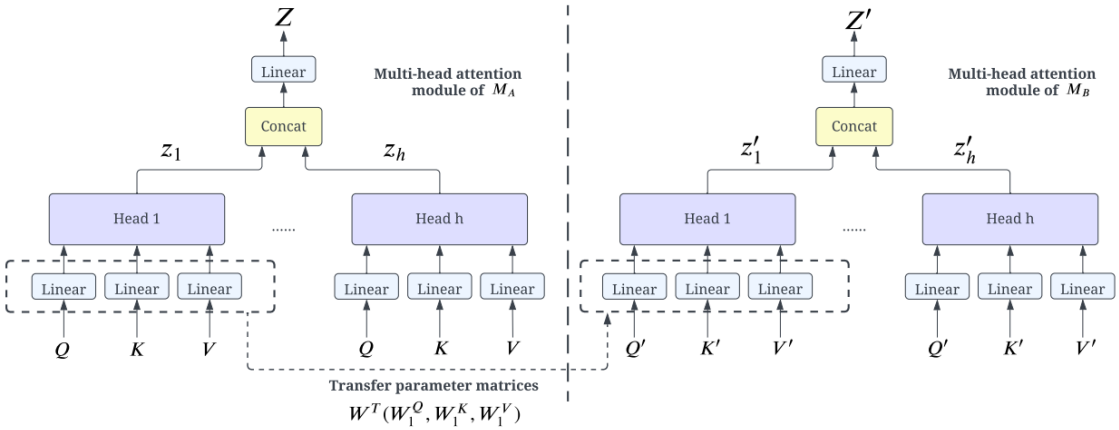

Figure 3. The knowledge transfer process. Taking head 1 as the transferable head, its parameter matrices of linear layers are transferred from M_A to M_B.

the target class. The classification result is based on *P* with a threshold of 0.5. The training target is to minimize the loss as equation 4.

$$Loss = -(y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})) \quad (4)$$

Where $\hat{y}$ is the probability that the model predicts the sample as a positive example; y is the label of the sample, which takes a value of 1 if the sample is positive, and 0 otherwise.

The trained *D* identifies which programming language *Z* is from. The rationale is that *D* captures the non-shared features between *A* and *B*. Since the output of multi-head attention comes from the weighted concatenate of each head, these non-shared features also come from the subspaces of heads. Non-transferable heads contribute more to non-shared features, helping *D* distinguish *A* from *B*; Transferable heads are the opposite. We use the F1 score to represent the predictive performance of *D* since it considers both precision and recall.

$$F1 = \frac{2Percision \cdot Recall}{Percision + Recall} \quad (5)$$

To evaluate the contribution of each head to *D*, we replace the concatenate of *h* different heads as *h* repeated heads in the multi-head attention module. Then the *Z* becomes Z_i for the *i*-th head as equation 6.

$$Z_i = Concat\left(\overbrace{z_i, z_i, \dots, z_i}^{h}\right) W^O \quad (6)$$

B. Transfer Process

In this way, the feature space of Z_i is totally contributed by the *i*-th head. Since *Z* has the same size as *Z*, it can be directly predicted by *D*. The strategy of selecting transferable heads is based on the F1 score of each head. Head with a lower F1 score means its output Z_i is more difficult to be predicted by *D*, i.e., this head more focuses on shared features. In this paper, we regard the F1 score of the multi-head module as a threshold: For heads with a lower F1 score than the threshold, we identify them as transferable heads.

The knowledge transfer process shows as Fig. 3. According to equation 1 and 3, the output z_i of the *i*-th head is calculated by linear transformations of input *Q*, *K*, *V*. Thus, the knowledge transfer is to transfer the parameter matrices W_i^Q, W_i^K, W_i^V of linear layers. Supposing there is an undertrained multi-head attention module of *B*, we transfer the linear layer of the *i*-th transferable head from *A* to *B*. Then the output z'_i of the transferred head is

$$z'_i = Attn\left(QW_i^Q, KW_i^K, VW_i^V\right)t_i \quad (7)$$

Where t_i is the transfer weight normalized from the F1 score of the *i*-th head.

After M_B receives *n* transferable heads from M_A, M_B is able to capture some common features between *A* and *B* as well. However, there are domain-specific features of *B* that need to be captured by M_B to fit the code summarization task in its domain. Thus, we freeze the transferred heads and train other components (W_i^O, feed-forward layers, and decoder) of M_B on its dataset. Supposing there are *n* transferred heads, the remaining *h* - *n* heads are trained from scratch to capture domain-specific features of *B*.

IV. Experiments and Discussion

A. Dataset and Baselines

We take Python as the source domain and Solidity as the target domain in our experiments. These two are both object-oriented programming languages, while Solidity is used in the blockchain area. Their shared features (types, object-oriented characteristics, etc.) and non-shared domain-specific features are significant, which is ideal for transfer learning.

We randomly selected 450K Python <*code*, *comment*> pairs from the widely used corpus CodeSearchNet [4], and 45K Solidity pairs from the corpus used in [6] to build our dataset. We split original code texts by a set of symbols, i.e., { . ,"~' : * () ! - (space)}. Each token written in camel-case or snake-case shall be segmented into the original word. For example, the token **helloWorld** or **hello_world** shall be segmented into two separate words: **hello** and **world**. On this basis, we build our vocabularies for sentence generation. We use four state-of-the-art Transformer-based code summarization model [1, 2, 5, 8] introduced in section II as baselines.

TABLE I. PERFORMANCE COMPARISON (IN PERCENTAGE) OF BASELINES WITH AND WITHOUT TRANSFER LEARNING.

Language	Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	Rouge-L	METEOR
Python	[1]	43.31	38.54	35.25	31.02	39.26	17.17
	[2]	40.17	35.42	31.09	26.34	33.67	14.18
	[5]	41.05	36.22	33.02	28.39	34.15	15.38
	[8]	39.39	35.75	34.87	27.69	38.66	16.54
Solidity	[1]	27.31	23.13	19.65	14.50	21.09	9.96
	[1]+TL	**33.50**	**30.37**	**28.26**	**25.20**	**26.45**	**13.19**
	[2]	24.49	21.72	18.37	12.66	18.63	7.09
	[2]+TL	**30.56**	**26.21**	**23.33**	**20.01**	**21.45**	**11.43**
	[5]	24.99	22.37	19.45	13.78	18.56	8.34
	[5]+TL	**31.87**	**28.08**	**23.76**	**18.44**	**21.96**	**10.85**
	[8]	22.18	17.46	14.12	10.49	17.30	9.24
	[8]+TL	**31.57**	**26.35**	**23.88**	**19.29**	**20.96**	**11.02**

a. Python is the source domain, and Solidity is the target domain. TL is the abbreviation of the proposed transfer learning approach.

B. Experimental Settings and Metrics

The word embedding size and multi-head attention size are both set to be 512. The number of heads *h* is set to be 16. The mini-batch size is set to be 64 with a learning rate of 0.001 for both Transformer-based models and the discriminator. We use 10-fold cross-validation for experiments and run 10 epochs in the training processes. All the experiments in this paper are implemented with Python 3.7 and run on Google Colab with an NVIDIA Tesla P100 GPU.

We evaluate the performance of code summarization task based on three widely used metrics, BLEU, ROUGE-L, and METEOR. These three objective metrics are close to human evaluation criteria. BLEU is obtained by calculating the n-gram matches between the candidate and reference sentences. We use BLEU 1-4 as our metrics as in [8]. ROUGE-L is a metric that matches the longest common sequence between two sentences and returns the recall rate. METEOR combines both uni-gram matching precision and recall rate using harmonic mean.

C. Experimental Results

Table I compares the effectiveness of proposed transfer learning approach between four baselines on our dataset. The transfer learning mechanism brings improvement of BLEU 1-4 ranged from 4.31% (BLEU-3 of [5]) to 10.70% (BLEU-4 of [1]); For ROUGE-L, the range of improvement is from 2.82% ([2]) to 5.36% ([1]); For METEOR, the range of improvement is from 1.78% ([8]) to 4.34% ([2]). Overall, the performance of the four baselines in the target domain has been significantly improved after leveraging transfer learning.

D. Threats to Validity

Similarity between programming languages. Transfer learning requires a high similarity between the data features of the source and target domains. Both Python and Solidity used in our experiments are object-oriented languages, so it is adequate to use transfer learning. It is not applicable to transfer knowledge between programming languages with significant differences in structure and syntax, such as an object-oriented language and an assembly language.

Number of transferable heads. Here is a trade-off: Fewer transferable heads mean less knowledge would be transferred; More transferable heads would reduce the model's ability to learn domain-specific features because there would be fewer trainable heads. For models with different numbers of heads, the ideal threshold of identifying transferable heads may vary according to experimental results.

V. CONCLUSION

We propose a transfer learning approach to accurately transfer knowledge between Transformer-based models for code summarization tasks. We train a discriminator to identify transferable heads that focus more on common features between source and target domains. Our approach only transfers knowledge in similar feature spaces between domains, which is more adaptive than simply copying and freezing neural layers. We conducted experiments on a dataset built from a large public corpus, proving the effectiveness of our approach.

REFERENCES

[1] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. A transformer-based approach for source code summarization. arXiv preprint arXiv:2005.00653, 2020.

[2] Colin B Clement, Dawn Drain, Jonathan Timcheck, Alexey Svyatkovskiy, and Neel Sundaresan. Pymt5: multi-mode translation of natural language and python code with transformers. arXiv preprint arXiv:2010.03150, 2020.

[3] Wu Hongqiu, Zhao Hai, and Zhang Min. Code summarization with structure-induced transformer. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL), 2021.

[4] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. CodeSearchNet challenge: Evaluating the state of semantic code search. arXiv preprint arXiv:1909.09436, 2019.

[5] Xin Liu and Liutong Xu. A combined model for extractive and abstractive summarization based on transformer model. In SEKE, pages 396–399, 2020.

[6] Chaochen Shi, Y ong Xiang, Jiangshan Y u, and Longxiang Gao. Semantic code search for smart contracts. arXiv preprint arXiv:2111.14139, 2021.

[7] Ashish V aswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.

[8] Ruyun Wang, Hanwen Zhang, Guoliang Lu, Lei Lyu, and Chen Lyu. Fret: Functional reinforced transformer with bert for code summarization. IEEE Access, 8:135591–135604, 2020.