Enhancing Code Retrieval through Deep Learning and Information Retrieval Fusion

Wenshuo Cheng[1,2,a], Jianbo Jiang[1,2,b], Junyu Lu[3,c*]

[1]AHU-IAI AI Joint Laboratory, Anhui University, Hefei, China
[2]Institute of Artificial Intelligence, Hefei Comprehensive National Science Center, Hefei, China
[3]School of Data Science, University of Science and Technology of China, China

[a]WA22301171@stu.ahu.edu.cn, [b]WA21301043@stu.ahu.edu.cn,
[c]lujunyu@mail.ustc.edu.cn

Abstract. Code retrieval is a widely used technique that can search for the most relevant code fragments based on developers' natural language queries. Most of the existing work feeds the whole code directly into the deep learning model for training, and does not effectively utilize auxiliary information such as method name or input parameter. In fact, the auxiliary information in the code is intuitive, easy to obtain, and can be very helpful for the improvement of retrieval results. In this paper, we summarize the code information into two categories, implicit structural information and explicit auxiliary information. To make more rational use of these two types of information, this paper proposes a two-stage code retrieval model. In the first stage, deep learning is used to mine the implicit structural information in the code. We adopted an improved code translation mechanism to recall multiple code segments. In the second stage, information retrieval is used to mine the explicit auxiliary information in the code, achieving a reranking of the recall results from the first stage. We validated our method on the Java dataset of CodeSearchNet. The experimental results prove that our method is effective and achieves good results.

Keywords: code retrieval, information retrieval, code translation

1 Introduction

In the era of large-scale code, software developers write code more often by searching for high-quality code on the Internet, such as Github or Stack Overflow, in order to improve the efficiency of code development [1]. As the process illustrated in Figure 1, given a query in natural language by a programmer, one code snippet returned as the most suitable code segment, code search technology aims to provide faster and more accurate search results.

Early code retrieval was primarily based on Information Retrieval (IR), treating code as ordinary text and matching query text with code text through keywords. In this regard, many works have been conducted to expand or reconstruct the matching content. For example, the model proposed by Lu et al. [2] uses synonyms generated by WordNet

T. Yao et al. (eds.), *Proceedings of the 2024 3rd International Conference on Engineering Management and Information Science (EMIS 2024)*, Advances in Computer Science Research 111,
https://doi.org/10.2991/978-94-6463-447-1_33

[3] to expand the query. The CodeHow model [4] uses related APIs to expand the query and searches for code by extending the Boolean model to use matched APIs and query keywords. However, information retrieval methods always have a significant drawback in that they can only stay at the level of text information and find it difficult to bridge the gap between programming languages and natural languages. In 2018, deep learning was first applied to the field of code retrieval in DeepCS [5]. There have also been more ways to represent code, such as representing code as an Abstract Syntax Tree (AST) or Control Flow Graph (CFG) in MMAN [6], and matching text representation graphs and code representation graphs using the method of Graph Convolutional Networks in DGMS [7]. However, due to the complexity of high-level programming languages, it is still very difficult to achieve very good results in code representation and learning of code semantics.

Regardless of the method, the extraction of code semantics is a crucial step. This paper summarizes the information in the code into the following two aspects. As shown in the Figure 2, the first is explicit auxiliary information, such as method names or input parameters which are highly related to code semantics and are easy to extract. The second is implicit structural information, such as code execution logic. Without relevant knowledge, it is difficult to understand the semantic information of high-level programming languages, so this type of information often requires preliminary code representation processing. However, existing work does not make a clear distinction between these two types of information. More consideration is given to the information of the entire code segment, and explicit auxiliary information often does not get effectively utilized.

Fig. 1. Code Retrieval Process

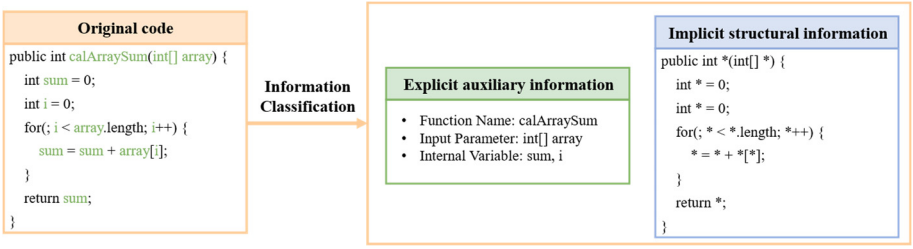


Fig. 2. Code Information Classification

In order to utilize the explicit auxiliary information and implicit structural information in the code more effectively, this paper proposes a Two-Stage Code Retrieval model named TS-CR. In the first stage, deep learning methods are used to match the implicit structural information in the code. In the second stage, information retrieval methods are used to match the explicit auxiliary information in the code. This approach

allows both types of information to be efficiently utilized. The specific practices are described in Section 2.

To summarize, this paper makes the following contributions:

- We propose a two-stage code retrieval model, TS-CR, enables both implicit structural information and explicit auxiliary information to be effectively utilized, ultimately enhances the effectiveness and interpretability of code retrieval.
- We have improved the code translation techniques to make the result of code translation more streamlined and the accuracy of code retrieval is improved.
- We evaluated our approach on queries in the CodeSearchNet corpus. The experimental results show that our proposed method is effective and can significantly improve the retrieval results.

2 Methodology

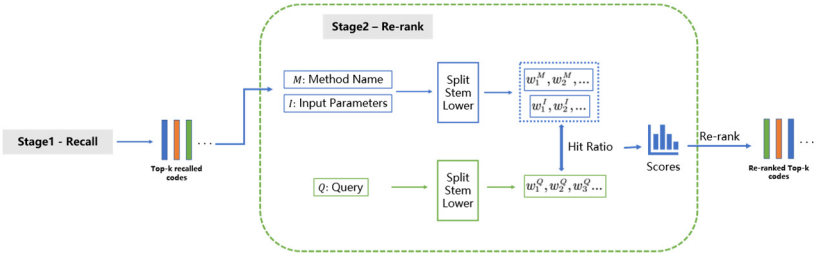


Fig. 3. Overview of the TS-CR model

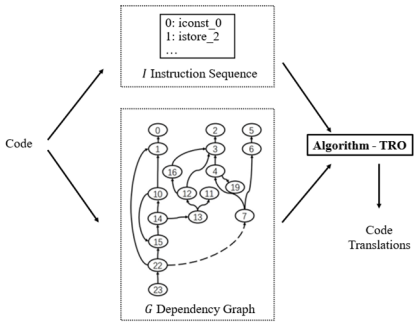
The Figure 3 provides an overview of the TS-CR model. In the first stage, deep learning model is used to mine implicit structural information and recall k segments of code, this k segments of code are re-ranked in the second stage using explicit auxiliary information. For the deep learning model in the first stage, we use TranCS [8] and make improvements in the instruction translation phase, the detailed process is described in Section 2.1. In the second stage, we will compute the similarity between the code auxiliary information and the query using our model (described in Section 2.2) in order to re-rank the codes. Finally, the re-ranked Top-k codes are returned.

2.1 Translation Rule Optimization

Code translation provide a detailed description of all the operations in the code, better preserving the semantics of the code. Therefore, we believe that using code translation techniques is a reasonable method, capable of bridging the significant semantic gap between natural language and code. The embedding of code translation and the query are then fed into their Encoder respectively for model training, making the vectors between the same pair of code and query closer. At the end of this stage, the trained model will recall Top-k codes and send them to the second stage.

However, the existing code translation work TranCS has some limitations. The results of code translation contain a lot of redundancy, which makes the translation results excessively long and affects the model's performance.

In response to the limitation, we designed the Algorithm - Translation Rule optimization shown in Figure 4.



Algorithm Translation Rule Optimization

Input: Instruction Sequence I; Dependency Graph G;
Output:Optimized Instruction Translation T;

```
set OptList = {}
for each pair (i, i + 1) in I do:
    set J = I – {i, i + 1}
    if (i, i + 1) ∈ G and forall j in J, (j, i + 1) ∉ G and (i, j) ∉ G then
        put (i, i + 1) in the set OptList
    end if
end for
for item in OptList do
    if item[0] ∈ 𝕀^POU then
        replace data destination in item[0] with data destination in item[1]
        delete item[1] in I
    else:
        replace data source in item[1] with data source in item[0]
        delete item[0] in I
    end if
end for
T = translate( the optimized I)
return T
```

Fig. 4. Translation Optimization Algorithm

As shown in Figure 4, $\mathbb{I}^{POU}$ denotes an instruction containing a computational operation. After this algorithm, many interdependent instructions can be merged with no loss of information, thus reducing the length of the final code translation.

2.2 Re-rank with Explicit Auxiliary Information

In the second stage, we re-rank the Top-k code segments with the explicit auxiliary information (named IR-rerank). We have implemented a detailed word processing method from the following two aspects:

1. Split Word: Words are generally made up of multiple words, so they need to be split.
2. Word Stem: Get the stem of each word to make them match each other.

As shown in the Figure 3, in the specific re-ranking process, after processing both the query text and code information, a word list $w_1, w_2, ...w_n$ is generated, we represent word list as a set, where M stands for method name, I for input parameters, and Q for query. The similarity is calculated as the following formula:

$$similarity = \frac{|M \cap Q|}{|M|} + \frac{|I \cap Q|}{|I|}. \quad (1)$$

The higher the similarity, the higher the ranking will be. If the two pieces of code have the same similarity, the ranking will be based on the first stage. Eventually the re-ranked codes are returned.

3 Evaluation

3.1 Dataset and Evaluation Metrics

Dataset.

In this paper, we evaluated our performance on the Java code from the CodeSearchNet (CSN) public corpus [9]. The baselines we considered for comparison include DeepCS [5], MMAN [6], and TranCS [8]. Regarding the data, we filtered and reallocated the original data from CSN, ultimately obtaining 69,324 samples as the training set and 1,000 samples as the test set.

Evaluation Metrics.

During model evaluation, we have 1000 test samples, where each query in the sample has a corresponding correct code and the remaining 999 codes are interference terms. We adopt two evaluation metrics widely used in retrieval research [10] to measure the performance of our model, the success rate at k ($SuccessRate@k$) and the mean reversal rank (MRR). The higher the MRR and $SR@k$ values, the better the code retrieval performance.

3.2 Evaluation Results

Overall Results.

Table 1. Overall Performance of TS-CR

Tech	$SR@1$	$SR@5$	$SR@10$	MRR
DeepCS	0.276	0.524	0.622	0.391
MMAN	0.335	0.562	0.657	0.436
TranCS	0.540	0.770	0.831	0.640
TranCS + TRO	0.542	0.775	0.842	0.654
TranCS + IR-rerank	0.600	0.809	0.831	0.691
TS-CR (ALL)	**0.603**	**0.811**	**0.842**	**0.697**

From the Table 1, it can be seen that experimental results have made considerable progress, both the TRO (Translation Rule Optimization) and IR-rerank collectively improve the performance of the TS-CR model, and the final MRR can be improved to 0.697, an increase of 0.057. This proves that our modeling is effective, TRO allows the code information to be represented more comprehensively, and the explicit auxiliary information is effectively utilized in the second stage.

Table 2. Effect of Two Messages

Tech	$SR@1$	$SR@5$	$SR@10$	MRR
TranCS	0.542	0.775	0.842	0.654

TranCS + FuncName	0.564	0.796	0.831	0.665
TranCS + InParam	0.556	0.770	0.831	0.656
TS-CR(ALL)	**0.603**	**0.811**	**0.842**	**0.697**

The Table 2 shows the different impacts of method names and input parameters on experimental results. It can be observed that when applied individually, the effects are rather ordinary. After numerous experiments, we find that the best results are achieved by directly adding the two types of similarity with equal weight, particularly when k is set to 10.

Table 3. Comparison of the Two Stages

Tech	$SR@1$	$SR@5$	$SR@10$	MRR
Only Stage 1	0.542	0.775	0.842	0.654
Only Stage 2	0.372	0.602	0.701	0.483
TS-CR(k=100)	0.556	0.784	0.855	0.658
TS-CR(k=10)	**0.603**	**0.811**	**0.842**	**0.697**

Analysis of Model Rationality.

In the model of this paper, the two stages are paired and applied, with the first stage recalling some codes and the second stage re-rank these codes. Table 3 shows the effect of the two stages when they are applied individually, as well as the effect of combining the two stages at different k values. As can be seen from the table, when applied alone, the results are worse when using only the second stage than when using only the first stage. This shows that the deep learning method has stronger recall ability and the information retrieval method is able to achieve a small range of word matching, and the

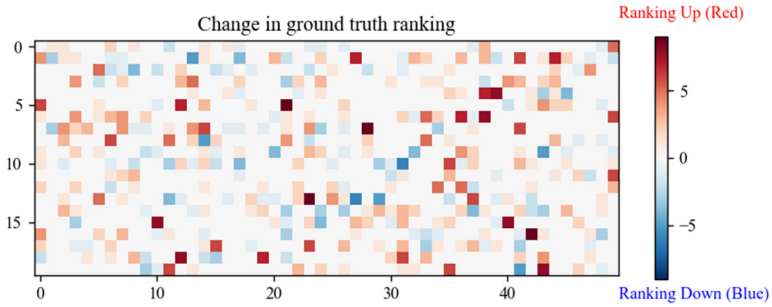


Fig. 5. Change in Ground Truth Ranking

two methods complement each other and promote each other to make TS-CR achieve better results.

The Table 3 also shows the effect at different values of k. The results show that the effect is much better when the k value is set to 10 than when k is 100, and according to our experiments, the highest value of MRR is only obtained when k=10, and the effect is worse at the rest of the time. This suggests that the combination of the two phases,

although effective, needs to be at the proper k value to be at its best and cannot be biased towards either stage.

The heatmap presented in Figure 5 illustrates the ranking changes of 1000 samples after undergoing the second stage. Red indicates an increase in the ranking of the ground truth, while blue indicates a decrease. It is evident that the number of red instances significantly exceeds the number of blue ones (with 219 samples in red and 129 in blue), thus, the heatmap adequately demonstrates the effectiveness of the second stage, with an approximate 22% of the samples showing a significant improvement in their ranking.

4 Conclusions

In this paper, we propose a two-stage code retrieval model, TS-CR, which captures both the implicit structural information and explicit auxiliary information of the code, ensuring that all information is efficiently utilized. Additionally, we have optimized existing code translation techniques to make the translation results more concise. Comprehensive experiments conducted on the CSN Java dataset demonstrate that TS-CR is an effective method of code retrieval, outperforming other existing works.

We believe that the two-stage code retrieval model is reasonable, which allows code information to be fully and effectively learned. In the future, we will try to apply our second stage to more other deep learning models to prove that this part of our work can make a wider contribution. At the same time, we will also explore more effective re-ranking methods.

References

1. Brandt J, Guo P J, Lewenstein J, et al. Two studies of opportunistic programming: interleaving web foraging, learning, and writing code[C]//Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. 2009: 1589-1598.
2. Lu M, Sun X, Wang S, et al. Query expansion via wordnet for effective code search[C]//2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER). IEEE, 2015: 545-549.
3. Miller G A. WordNet: a lexical database for English[J]. Communications of the ACM, 1995, 38(11): 39-41.
4. Lv F, Zhang H, Lou J, et al. Codehow: Effective code search based on api understanding and extended boolean model (e)[C]//2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2015: 260-270.
5. Gu X, Zhang H, Kim S. Deep code search[C]//Proceedings of the 40th International Conference on Software Engineering. 2018: 933-944.
6. Wan Y, Shu J, Sui Y, et al. Multi-modal attention network learning for semantic source code retrieval[C]//2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2019: 13-25.
7. Wan Y, Shu J, Sui Y, et al. Multi-modal attention network learning for semantic source code retrieval[C]//2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2019: 13-25.

8. Sun W, Fang C, Chen Y, et al. Code search based on context-aware code translation[C]//Proceedings of the 44th International Conference on Software Engineering. 2022: 388-400.
9. Husain H, Wu H H, Gazit T, et al. Codesearchnet challenge: Evaluating the state of semantic code search[J]. arXiv preprint arXiv:1909.09436, 2019.
10. Cambronero J, Li H, Kim S, et al. When deep learning met code search[C]//Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2019: 964-974.