R2PS: Retriever and Ranker Framework with Probabilistic Hard Negative Sampling for Code Search

Anonymous ACL submission

Abstract

Code search, which retrieves relevant codes for given user queries, can greatly enhance the productivity of programmers. The typical method for this task employs a dual-encoder architecture to encode semantic embeddings of query and code, and then calculate their similarity based on the embeddings. However, this architecture cannot model token-level interactions between query and code, limiting the code search performance. To address this limitation, we introduce a cross-encoder architecture for code search, which learns the semantic matching of query and code through joint encoding.

Despite its effectiveness, the cross-encoder, which needs to encode the concatenations of all codes and a given query for online serving, is computationally infeasible. To address this issue, we introduce a Retriever-Ranker (RR) framework for the code search task, comprising a dual-encoder retriever that mines the most relevant k codes for a given query, and a cross-encoder to further rank the retrieved codes. Furthermore, we address the issues of discrepancy and false negatives that can occur during training of the cross-encoder in the RR framework through the use of a probabilistic hard negative sampling strategy. Experiments on four datasets using three code encoders (CodeBERT, GraphCodeBERT, UniXcoder) demonstrate the superiority of our proposed method.

1 INTRODUCTION

Code search aims to retrieve the relevant codes from a large collection of codes for given queries described by natural language (Gu et al., 2018; Cambronero et al., 2019; Ling et al., 2021). The core of code search is to predict whether queries are relevant to codes. Traditionally, classical information retrieval (IR) methods such as term frequency-inverse document frequency (TF-IDF) have been widely adopted for code search (Diamantopoulos et al., 2018). Specialized rules are often designed to extract features of codes (Luan et al., 2019), and term frequencies are exploited to identify relevant queries and codes (Sisman and Kak, 2013). However, traditional IR methods have some inherent limitations; for instance, the rules designed are hard to cover complex and implicit features and the term mismatch issue can limit the ability to discover useful codes.

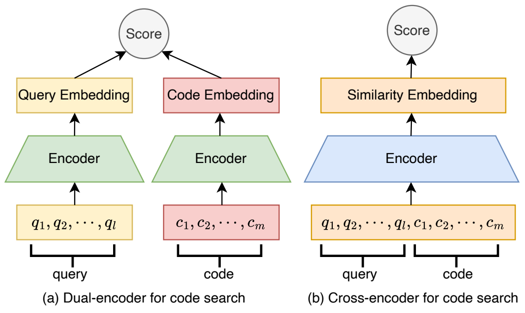

Figure 1: The dual-encoder architecture and the cross-encoder architecture for code search. (a) In dual-encoder, we input the token sequence of the query and code separately; (b) In cross-encoder, we input the token sequence concatenation of the query and the code.

Recently, deep learning techniques, particularly pre-trained language models (PLMs), have demonstrated remarkable success in code search, thanks to their ability to learn complex and implicit features (Gu et al., 2018; Guo et al., 2021). These methods typically employ a dual-encoder architecture, as depicted in Figure 1(a), which encodes query and code separately to acquire their embeddings as semantic representations, and then determines the similarity based on these embeddings (Cambronero et al., 2019).

The dual-encoder architecture fuses the information of query and code at the representation level, but this approach is unable to capture token-level interactions (Guo et al., 2016; Xiong et al., 2017) between query and code, thereby limiting the model's potential for optimal performance. To overcome this challenge, we propose a cross-encoder architecture for the code search task, as illustrated in Figure 1(b), This approach first concatenates the query and code and then encodes the concatenated query and code together. By doing so, the cross-encoder can model token-level interactions between query and code, which results in a more fine-grained understanding of the matching of query and code and a stronger model capability.

Although effective, the cross-encoder architecture has significant efficiency issues for online service. It requires concatenating each query with all codes in the codebase and encoding all of the concatenations, which becomes infeasible when dealing with large codebases containing millions of code snippets. On the contrary, the dual-encoder architecture is more efficient as it can encode all codes in the codebase offline, and only needs to encode the query online to find relevant codes based on the embeddings. Inspired by previous work on other information retrieval tasks such as question answering and text retrieval (Qu et al., 2021; Zhang et al., 2022), we introduce a retriever-ranker framework for code search that combines the advantages of the effectiveness of the cross-encoder and the efficiency of the dual-encoder. The framework consists of a dual-encoder as the retriever module and a cross-encoder as the ranker module. The retriever module retrieves a small set of relevant codes for the query, and the ranker module then further ranks this small set of codes.

Now, we identify two issues with the training of the cross-encoder in the RR framework: (1) Discrepancy problem: During inference, the ranker only needs to rank relevant codes retrieved by the retriever, but training with a random sampling strategy may include irrelevant codes as negative samples, causing a discrepancy between training and inference. (2) False negative problem: Unlabeled data may be positive, but the random sampling strategy may treat it as negative, harming the model training. To address these issues, we propose using a probabilistic hard negative sampling strategy to sample negative codes that are relevant to the query but not the most relevant. Specifically, we remove codes with low similarities to avoid the discrepancy and high similarities to avoid the false negative problem according to the well-trained dual-encoder, rescale the relevance scores of the remaining codes to sampling probability, and then sample negative codes according to this probability to train the cross-encoder. In this way, we can alleviate the discrepancy and false negative problems in the RR framework, and improve performance.

We name our proposed method as *Retriever and Ranker with Probabilistic Hard Negative Sampling method (R2PS)*, which uses a dual-encoder as the retriever and a cross-encoder as the ranker, and trains the cross-encoder using a probabilistic hard negative sampling strategy.

In summary, the main contributions of our paper are as follows:

- Highlighting the limitations of the dual-encoder architecture and proposing the use of the cross-encoder architecture for the code search task.
- Introducing the RR framework to code search, which combines the advantages of both cross-encoder and dual-encoder architectures for improved effectiveness and efficiency.
- Suggesting the use of probabilistic hard negative sampling to train the cross-encoder in order to address issues of discrepancy and false negatives.
- Conducting comprehensive experiments on four datasets using three code encoders, and demonstrating the superiority of our proposed R2PS.

2 RELATED WORK

2.1 Code Search

The core of code search is to predict whether queries are relevant to codes (Gu et al., 2018). To determine the relevance score, traditional information retrieval (IR) methods rely on rule-based code search, such as term matching (e.g., term frequency-inverse document frequency, TF-IDF), and manually-designed features based on the analysis of the abstract syntax tree of codes (Luan et al., 2019; Sisman and Kak, 2013; Diamantopoulos et al., 2018). However, these methods lack the ability to understand semantic information and have some limitations: (1) The term mismatch problem limits the ability to identify potentially useful codes; (2) Manually-designed features are inadequate for capturing complex and implicit features.

Recently, deep learning methods have been applied to the code search task with great success due to their ability to model semantic information and extract implicit features (Cambronero et al., 2019; Sachdev et al., 2018). Typically, the dual-encoder architecture is used for code search, which maps the query and code to embeddings using neural encoders, and calculates the similarity based on the embeddings (Cambronero et al., 2019). Various types of encoders have been used, such as

LSTMs (Wan et al., 2019; Hochreiter and Schmidhuber, 1997), Graph Neural Networks (Ling et al., 2021; Kipf and Welling, 2017), and Transformer-based encoders (Feng et al., 2020; Vaswani et al., 2017). Among these encoders, Transformer-based encoders with pre-training techniques, known as Pre-trained Language Models (PLMs) in some literature (Bui et al., 2021; Devlin et al., 2019), have recently been focused for their effectiveness (Li et al., 2022a; Niu et al., 2022). Transformer-based encoders treat both query and code as token sequences and map them to embeddings for semantic matching. Many pre-training tasks have been proposed to pre-train code-relevant Transformer-based encoders, and some pre-trained models (Wang et al., 2022; Lachaux et al., 2021) have been released as foundation models to initialize models for downstream tasks such as code search. In this paper, we adopt three typical code-relevant PLMs, CodeBERT (Feng et al., 2020), GraphCodeBERT (Guo et al., 2021), and UniXcoder (Guo et al., 2022), as foundation models to validate the effectiveness of our proposed method. Additionally, some work concatenates the query and code and then encodes them together to learn relevance score (Shi et al., 2022; Chai et al., 2022). Our work is different from them these methods in the following ways: (1) We use the RR framework to integrate the dual-encoder and cross-encoder to achieve an effective and efficient result. (2) We use the probabilistic hard negative sampling strategy to mitigate the training-inference discrepancy and false negative problems.

2.2 Retriever and Ranker Framework

The retrieval and ranker framework is widely used in information retrieval areas, including question answering (Qu et al., 2021), text and document retrieval (Ren et al., 2021; Zhang et al., 2022). The retriever is faster than the ranker module, while the ranker is more precise than the retriever (Chen et al., 2017). In these areas, some works propose using the traditional IR methods as the retriever and the dual-encoder as the ranker (Chen et al., 2017). Later, some works propose using a dual-encoder for the question (query) and answer (text/document) as the retriever and a cross-encoder for the concatenation of question (query) and answer (text/document) as the ranker (Soldaini and Moschitti, 2020; Zhang et al., 2022). However, to the best of our knowledge, no previous work has introduced the retrieval and ranker framework to the code search task. Thus, our work is the first to introduce the RR framework to the code search task, and we also demonstrate its effectiveness through extensive experiments.

3 METHOD

In this section, we present R2PS, a retriever and ranker framework with a probabilistic hard negative sampling method for code search. We begin by introducing the code search task and the conventional dual-encoder architecture for code search. We then describe the cross-encoder architecture, the retrieval and ranker framework for the code search task, and finally, introduce the probabilistic hard negative sampling method for training the cross-encoder within the RR framework.

3.1 Task Formulation and Dual-encoder

Assuming we have a code corpus containing various code snippets c_i, where $i = 1, 2, \cdots, N$, each implementing a specific function. Given a user query q, described in natural language, the objective of code search is to identify and present a limited number of relevant codes to the user based on relevance scores (also known as similarities). The core of code search is to estimate the relevance scores of queries and codes.

As depicted in Figure 1(a), the dual-encoder architecture is commonly used to predict relevance scores for code search. The query encoder first maps the query token sequence to a query embedding, and the code encoder maps the code token sequence to a code embedding. The query encoder and code encoder can be the same and represented as $E(\cdot)$. The similarity between the query and the code can be calculated using the dot-product:

$$s_{dual}(q, c) = E(q) \cdot E(c). \tag{1}$$

3.2 Cross-encoder and RR Framework

We introduce a cross-encoder architecture to the code search task to enhance model capability compared to a dual-encoder. Additionally, we introduce a RR framework that incorporates the effectiveness of the cross-encoder with the efficiency of the dual-encoder for code search.

Cross-encoder: To effectively model interactions between query and code at the token level, we propose using a cross-encoder to calculate the relevance score, as shown in Figure 1(b). First,

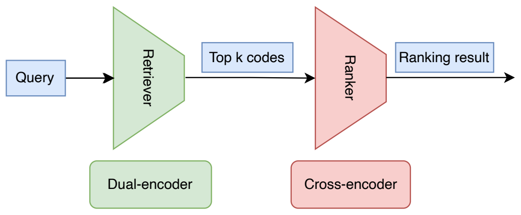

Figure 2: Our proposed Retriever and Ranker framework for code search.

we concatenate the query token sequence and code token sequence to a unified sequence $[q, c]$. Then, we use the encoder to map the unified sequence into an embedding. The encoder is the same as the one used in the dual-encoder, such as a transformer encoder, but with a longer input sequence length. Finally, using a neural network head $H(\cdot)$, we map the embedding to a scalar to represent the relevance score of the query and code. The cross-encoder architecture can be represented as:

$$s_{cross}(q, c) = H(E([q, c])). \qquad (2)$$

Retriever and Ranker Framework: To take advantage of both the effectiveness of the cross-encoder and the efficiency of the dual-encoder, we propose a retriever and ranker framework for online code search service, as illustrated in Figure 2. First, the dual-encoder is employed as the retriever to identify the top k most relevant codes for a given user query. Next, the cross-encoder is used as the ranker to rank the k codes and present them to the user in sequence.

For the dual-encoder, we can calculate the code embeddings for the entire codebase during pre-processing and represent them as $E(c_i), i = 1, 2, \cdots, N$. This means that during online serving, the retriever only needs to calculate the embedding of the given query $E(q)$ and find the k most similar codes based on the pre-calculated code embeddings. The cross-encoder then calculates the similarities between the query q and all k codes and ranks them accordingly. As a result, the online encoding time for a query only involves one forward propagation of the dual-encoder for the query and k forward propagations of the cross-encoder for the concatenations of the query and k codes, which is independent of the number of codes in the codebase. By increasing k, the dual-encoder finds more relevant codes for the cross-encoder to rank, resulting in better performance; With a smaller k, the cross-encoder performs fewer propagations, reducing the computing cost. Thus, the proper value for k can balance performance and computing cost.

3.3 Model Training

To mitigate the issues of discrepancy between training and inference and false negatives in the RR framework, we propose a probabilistic hard negative sampling method for training the cross-encoder. We will begin by discussing the InfoNCE loss (van den Oord et al., 2018; He et al., 2020) and explain how to train the dual-encoder using it. Next, we will introduce our proposed probabilistic hard negative sampling method for constructing negative samples to train the cross-encoder.

The goal of training a code search model is to learn a relevance estimation function that assigns higher scores to pairs of queries and codes that are relevant to each other than to those that are not. Given a query q and its corresponding relevant code c^+ as positive code, and a set of non-relevant codes $\{c_i^-\}_{i=1}^m$ as negative codes, the InfoNCE loss can be formulated as below:

$$\mathcal{L}_q = -log \frac{e^{s(q,c^+)/\tau}}{e^{s(q,c^+)/\tau} + \sum_{i=1}^m e^{s(q,c_i^-)/\tau}}, \qquad (3)$$

where $s(q, c)$ denotes the relevance score of the query-code pair estimated by the neural network model $s(\cdot)$, τ denotes the temperature hyperparameter. Minimizing this loss will increase the relevance score of the paired query-code $s(q, c^+)$ and decrease the relevance scores of the query with non-relevant codes $s(q, c_i^-)$.

The question now is how to sample the negative codes $\{c_i^-\}_{i=1}^m$. One common method is in-batch negative sampling, which is widely used to optimize the dual-encoder architecture due to its reusability of the intermediate embedding representation and lower training complexity. Given a batch of paired data $\{(q_j, c_j)\}_{j=0}^m$, this strategy uses all unpaired codes in the batch as negative codes for each query q_j, that is, $\{c_i\}_{i=0}^m$ and $i \neq j$. We also use the strategy to train the dual-encoder in our RR framework.

To train the cross-encoder, we propose a probabilistic hard negative sampling strategy to mitigate the issues of training-inference discrepancy and false negative samples, as shown in Figure 3. Using the well-trained dual-encoder, we can calculate the similarities of the query q with all codes $\{c_i\}_{i=1}^N$ in the codebase. We eliminate two types of codes for improved negative sampling:

- At the inference stage, the cross-encoder is only responsible for ranking the relevant codes. To

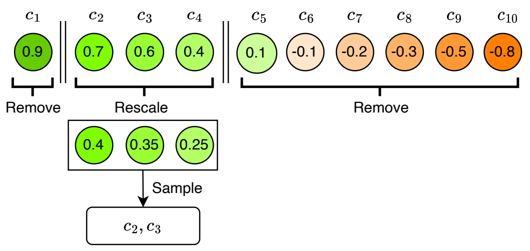

Figure 3: Our proposed probabilistic hard negative sampling method.

reduce the discrepancy between training and inference, we remove irrelevant codes whose similarities are below a threshold ρ_1.

- False negatives refer to query-code pairs that may be relevant but have not been labeled as such. Codes with high similarities are likely to be false negatives. To mitigate this problem, we remove codes whose similarities are above a threshold ρ_2, in order to reduce the number of false negatives.

The number of irrelevant codes is much greater than the number of false negatives, so the remaining codes are still relatively hard negative samples. We refer to these remaining codes as $\{c_i^*\}_{i=1}^{L}$, satisfying $\rho_1 < s_{dual}(q, c_i^*) < \rho_2$. Next, we rescale the similarities as a probability distribution by the following formula:

$$p_i = \frac{e^{s_{dual}(q,c_i^*)/\tau'}}{\sum_{j=1}^{L} e^{s_{dual}(q,c_j^*)/\tau'}}, i = 1, 2, \cdots, L, \quad (4)$$

and sample m negative samples according to this probability distribution as negative samples. Here, τ' adjusts the smoothness of the distribution (larger τ' results in a smoother distribution), and thus adjusts whether to sample more codes that are more relevant according to the dual-encoder. After sampling m negative codes, we can calculate the InfoNCE loss function and optimize the cross-encoder model.

4 EXPERIMENTS

In this section, we conduct experiments on four datasets to evaluate the performance of our proposed R2PS. The experiments are designed to address the following research questions:

- **RQ1:** Does our proposed R2PS surpass the performance of current state-of-the-art methods?
- **RQ2:** Is our proposed RR code search framework universal to be applied for different code search encoders?
- **RQ3:** Is R2PS suitable for real-time serving? Can we achieve a balance between effectiveness and efficiency by varying the number of retrieved codes in the RR framework?
- **RQ4:** Do all components in R2PS contribute to the overall performance?
- **RQ5:** Can our R2PS result in a more rational ranking distribution for the paired codes?

Table 1: Statistics of the datasets.

Dataset	Training	Validation	Test	Codebase
CSN-Ruby	24,927	1,261	1,400	4,360
CSN-JS	58,025	3,291	3,885	13,981
CSN-Go	167,288	7,325	8,122	28,120
CSN-Python	251,820	13,914	14,918	43,827
CSN-Java	164,923	5,183	10,955	40,347
CSN-Php	241,241	12,982	14,014	52,660
CosQA	19,604	500	500	6,267
AdvTest	251,820	9,604	19,210	-
StaQC	118,036	14,755	14,755	29,510

4.1 Experimental Setup

To validate the effectiveness of our proposed R2PS, we primarily utilize UniXcoder (Guo et al., 2022), the most powerful released code search encoder to our best knowledge, as the foundation model. In **RQ2**, we also use CodeBERT (Feng et al., 2020) and GraphCodeBERT (Guo et al., 2021) as the foundation models to validate the universality of the RR framework for the code search task.

Datasets. We evaluate using four code search benchmark datasets: CodeSearchNet (CSN) (Husain et al., 2019; Guo et al., 2021), AdvTest (Lu et al., 2021), StaQC (Yao et al., 2018), and CosQA (Huang et al., 2021). CSN is collected from GitHub and uses the function comments as queries and the rest as codes. It includes six separate datasets with various programming languages including Ruby, JavaScript, Go, Python, Java, and Php. AdvTest is a challenging variant of CSN-Python that anonymizes function and variable names. StaQC is obtained from StackOverFlow with question descriptions as queries and code snippets in answers as codes. CosQA is a human-annotated dataset with queries collected from Bing search engine and candidate codes generated from a well-trained CodeBERT encoder. Table 1 summarizes the dataset statistics, including the number of query-code pairs in the training, validation, and test set, and the number of codes in the codebase used for evaluation. The codebase for AdvTest is comprised of all codes in the validation and test sets respectively.

Evaluation Metrics. To evaluate the performance of code search, we use Mean Reciprocal Rank (MRR), a commonly used metric for code search in previous research (Guo et al., 2022). MRR calculates the average reciprocal rank of the paired codes for all queries. It is defined as:

$$MRR = \frac{1}{|D|} \sum_{(q,c)\, in\, D} \frac{1}{rank_c^{(q)}}, \tag{5}$$

where $rank_c^{(q)}$ is the rank of the paired code c among all code in the codebase for the query q, and $|D|$ is the total number in the dataset.

Baselines. The primary method we compare our proposed method to is UniXcoder (Guo et al., 2022) for two reasons: (1) UniXcoder is currently the most advanced method for the code search task among all released code search encoders, including CodeBERT (Feng et al., 2020), GraphCodeBERT (Guo et al., 2021), and SyncoBERT (Wang et al., 2021). (2) We use UniXcoder as the foundation model for our R2PS experiments, making the comparison between UniXcoder and R2PS fair to validate the effectiveness of our proposed method. In addition, we also compare our method to the two latest methods, CodeRetriever (Li et al., 2022a) and SCodeR (Li et al., 2022b) which pre-train the Transformer-encoder on contrastive loss. The results of CodeBERT, GraphCodeBERT, and UniXcoder are reproductions of the original work using the released codes, while the results of CodeRetriever, SCodeR, and SyncoBERT are directly taken from the SCodeR work (Li et al., 2022b).

Implementation Details. We follow the same experimental settings as UniXcoder (Guo et al., 2022), including 10 training epochs, a learning rate of 2e-5 with linear decay, and 0 weight decay. We set the temperature hyper-parameter τ in the InfoNCE loss function to 0.05 for CSN and CosQA datasets, and 0.025 for AdvTest and StaQC. The number of retrieved codes in the RR architecture is 10. For the CosQA dataset, we use the average of the dual-encoder and cross-encoder as the ranker module. In the InfoNCE loss, we set the number of negative samples to 31 to fully utilize the GPU memory. In probabilistic hard negative sampling, we set ρ_1 and ρ_2 to keep only a small set of samples beginning from rank 1 by the dual-encoder for CSN, CosQA, and AdvTest without tuning, and a small set of samples beginning from the top 0.4% rank for StaQC. We set the sampling hyper-parameter $1/\tau'$ as 0 without tuning. In fact, the default setting without tuning for most hyper-parameters can achieve satisfying performances in our method. However, the hyper-parameters also keep the flexibility to perform well for some rare data distributions. All experiments are conducted in a server consisting of 8 NVIDIA A100-SXM4-80GB GPUs. We will release our code and well-trained model when our paper is published.

Table 2: Performance comparison of different methods in terms of MRR (%). The last row denotes the improvement of R2PS relative to UniXcoder.

Method	CSN	CosQA	StaQC	AdvTest
CodeBERT	70.3	66.9	21.4	34.9
GraphCodeBERT	71.7	68.7	20.5	39.1
SyncoBERT	74.0	-	-	38.1
CodeRetriever	75.3	69.6	-	43.0
SCodeR	76.7	**74.5**	-	45.5
UniXcoder	74.4	69.6	25.8	42.5
R2PS	**79.2**	72.3	**29.4**	**47.0**
Improvement	4.8	2.7	3.6	4.5

Table 3: Performance comparison of six languages of CodeSearchNet.

Method	Ruby	JS	Go	Python	Java	Php
CodeBERT	68.8	62.7	89.0	68.5	68.4	64.2
GraphCodeBERT	69.4	65.4	90.3	70.8	70.0	64.1
SyncoBERT	72.2	67.7	91.3	72.4	72.3	67.8
CodeRetriever	75.3	69.5	91.6	73.3	74.0	68.2
SCodeR	77.5	72.0	92.7	74.2	74.8	69.2
UniXcoder	73.9	68.5	91.5	72.5	72.6	67.6
R2PS	**79.1**	**73.8**	**93.4**	**78.3**	**78.2**	**72.4**
Improvement	5.2	5.3	1.9	5.8	5.6	4.8

4.2 Performance Comparison with SOTA Methods (RQ1)

The performance of compared methods in terms of MRR is shown in Table 2, with the last row denoting the improvement of R2PS compared to UniXcoder. The CSN results are an average of the six languages in the dataset, and the specific results for the six languages can be found in Table 3. From these tables, we can observe the following: (1) Compared to UniXcoder, our R2PS shows a significant improvement (about 3.9 on average) in terms of MRR, demonstrating the effectiveness of our proposed R2PS. Additionally, R2PS consistently outperforms UniXcoder in all datasets across different scenarios and programming languages, showing the generalization capability of our proposed R2PS. (2) Overall, our R2PS performs better than the latest CodeRetriever and SCodeR methods. On the CSN and AdvTest datasets, R2PS

Table 4: Performance comparison of the dual-encoder architecture and the RR architecture with different encoders in terms of MRR (%).

Method	CSN	CosQA	StaQC	AdvTest
CodeBERT	70.3	66.9	21.4	34.9
+RR	75.9	68.3	26.2	42.0
GraphCodeBERT	71.7	68.7	20.5	39.1
+RR	77.0	69.8	24.3	45.4
UniXcoder	74.4	69.6	25.8	42.5
+RR	77.5	71.5	29.0	45.4

Table 5: The response time comparison of the dual-encoder, the cross-encoder, and RR framework with different scales of the codebase.

Method	1,000	10,000	100,000
Dual-encoder	7.3ms	7.9ms	17ms
Cross-encoder	4.9s	47s	469s
RR	58ms	61ms	79ms

outperforms SCodeR by a 2.5 margin, which is significant. Although R2PS does not outperform SCodeR on the CosQA dataset, we attribute this to the pre-training — the pre-trained model SCodeR is better than UniXcoder, the foundation model of our R2PS in the experiments of this paper. However, our R2PS can be used with many different pre-trained models (encoders) as shown in **RQ2** and we believe it will outperform SCodeR if we use SCodeR as the foundation model.

4.3 Performance Comparison with Various Encoders (RQ2)

We conduct experiments to investigate the universality of our proposed RR framework with various encoders. Specifically, we use three different pre-trained language models, CodeBERT, GraphCodeBERT, and UniXcoder, as foundation encoders and evaluate the performance of the dual-encoder and the RR framework. The results are shown in Table 4. In this table, the first line in each pair shows the result of the dual-encoder, and the second line shows the result of the RR architecture. From Table 4, we can see that the RR framework significantly improves the performance of the three encoders, which demonstrates the universality of our proposed RR framework — it is effective for many different code search encoders.

4.4 Efficiency Evaluation (RQ3)

In this section, we compare the computing cost of the dual-encoder, the cross-encoder, and our

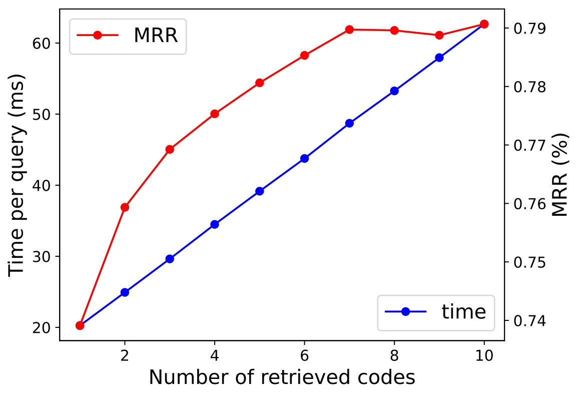

Figure 4: The computing cost and performance with the different number of retrieved code k.

RR framework during the online serving stage. Specifically, we simulate the response processing for incoming queries in online serving, where the server receives a user query and takes some time to infer the relevant codes and return them to the user. The code embeddings are calculated in the pre-processing stage, and the response time does not include the time to embed codes. We conduct experiments with three different codebase scales (1,000, 10,000, 100,000 codes) and calculate the average response time for 100 query requests on one GPU. The results are shown in Table 5, and we can observe that: (1) The cross-encoder takes around 8 minutes to respond for a codebase of 100,000 codes, which is too long for online serving. Therefore, the cross-encoder is almost impossible to implement in practice. (2) The response time of RR is about 79ms for a codebase of 100,000 codes, which is acceptable for online serving (no more than 100ms), making our R2PS usable in practice. (3) Overall, the response time of RR is longer than the dual-encoder, because the RR framework uses a ranker module to improve performance, which takes extra time. (4) As the codebase scale increases by 10 times, the computing time for the cross-encoder increases about 10 times, while the computing time for the dual-encoder and R2PS increases slowly (much less than 10 times). This is because for the cross-encoder, we have to encode the concatenation of the given queries and all codes in the codebase during online serving, while for the dual-encoder and RR framework, we can calculate the embeddings of all codes in the codebase in the data pre-processing stage.

Next, we conduct experiments by varying the number of retrieved codes k and show how it affects both effectiveness and efficiency as shown in

Table 6: The performance of ablation study.

Method	CSN	CosQA	StaQC	AdvTest
UniXcoder	74.4	69.6	25.8	42.5
+RR	77.5	71.5 (*68.8*)	29.0	45.4
+PS (R2PS)	79.2	72.3 (*70.4*)	29.4 (*24.3*)	47.0

Figure 4. Here we only report results about CSN-Ruby due to page limitation, and the results are similar for other datasets. From Figure 4, we can observe that: (1) Although the model performs better with a larger number of retrieved codes k, the rate of improvement decreases as k increases and performance does not improve much when $k > 7$. This suggests that performance will not continue to improve with increasing k and will converge to a maximum value. (2) The cost time increases linearly with k. This is because the cross-encoder needs k forward propagation, and the computation requirement increases linearly with k. Therefore, the model does not perform well for small k and costs too much time for large k. We can adjust a proper k value to achieve a balance between performance and acceptable computing time. Furthermore, using a large value for k (greater than 20) is not recommended for two reasons: the high cost of computation, making it difficult to provide real-time responses to users' queries, and the lack of a significant improvement in performance.

4.5 Ablation Study (RQ4)

We conduct an ablation study, as shown in Table 6, to examine the usefulness of both the RR architecture and probabilistic hard negative sampling (PS) components. The values in the table without brackets and the values in the brackets represent the results obtained using default hyper-parameters, while the values before the brackets represent the results obtained using tuned hyper-parameters. From this table, we can conclude: (1) Overall, both the RR framework and probabilistic hard negative sampling can enhance performance for the four datasets, thus confirming the effectiveness of these two strategies. (2) The model under default hyper-parameters does not perform well on the CosQA and StaQC datasets (as indicated by the results in the brackets), however, when the hyper-parameters are tuned, the model can achieve good results (as indicated by the results out of the brackets). This shows that the ability to adjust hyper-parameters offers flexibility in optimizing the model's performance on rare data distributions.

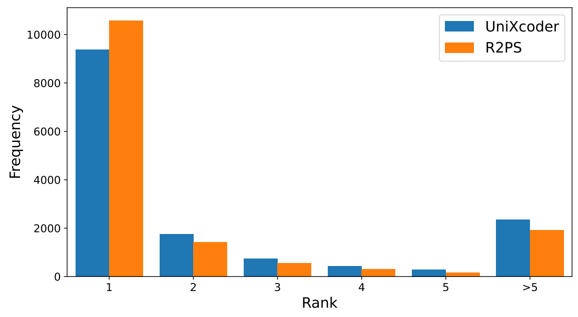

Figure 5: The distribution of the ranks of paired codes.

4.6 Rank Distribution Analysis (RQ5)

A search engine is a tool for filtering information and providing potentially useful information but it cannot guarantee accuracy. Therefore, users should take the time to understand it and verify its correctness. Users often scan search results from top to bottom, making it important to rank accurate information in the top position to save time. Codes, which can be complex and detailed, are not easy to understand completely. Therefore, it is crucial to rank and display the relevant codes in the top position, rather than just within the top k positions. In this study, we investigate whether our R2PS system is able to rank more relevant codes at the top position. Figure 5 illustrates the rank distribution of paired codes in the CSN-Python dataset. It can be observed that R2PS ranks more paired codes at the top position, thus saving users' time and improving satisfaction. Additionally, it can be seen that UniXcoder is able to rank about 80% of paired codes within the top 5 positions, which also provides the rationality of our RR framework: The dual-encoder is able to rank most paired codes within the top k, and the cross-encoder only needs to rank the top k codes to place the paired codes in the top positions.

5 CONCLUSION

In this paper, we present a retriever and ranker framework that utilizes a probabilistic hard negative sampling strategy to improve the performance of code search. Our paper makes three key contributions: the use of a cross-encoder to enhance model capabilities, a retriever and ranker framework to balance effectiveness and efficiency, and the use of probabilistic hard negative sampling to address issues with training-inference discrepancy and false negatives. We also conducted thorough experiments to demonstrate that our R2PS system can significantly improve performance with an acceptable increase in computing cost.

6 LIMITATION

Our paper mainly has the following limitations:

(1) Due to computational resource constraints, we only conducted experiments on three commonly used pre-trained models: CodeBERT, GraphCodeBERT, and UniXcoder. Further experimentation using other encoders and pre-trained models, such as CodeRetriever (Li et al., 2022a) and SCodeR (Li et al., 2022b), is left as future work.

(2) The datasets used in our paper contain a variety of biases: CSN and AdvTest consider comments as queries, CosQA suffers from exposure bias caused by previous models, and StaQC employs code snippets that may not convey full context. Thus, these datasets do not accurately represent real-world scenarios. Addressing the biases present in existing datasets is a worthwhile area of exploration for code search.

References

Nghi D. Q. Bui, Yijun Yu, and Lingxiao Jiang. 2021. Self-supervised contrastive learning for code retrieval and summarization via semantic-preserving transformations. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, pages 511–521. ACM.

José Cambronero, Hongyu Li, Seohyun Kim, Koushik Sen, and Satish Chandra. 2019. When deep learning met code search. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, pages 964–974. ACM.

Yitian Chai, Hongyu Zhang, Beijun Shen, and Xiaodong Gu. 2022. Cross-domain deep code search with meta learning. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*, pages 487–498. ACM.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1870–1879. 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.

Themistoklis Diamantopoulos, Georgios Karagiannopoulos, and Andreas L. Symeonidis. 2018. Codecatch: Extracting source code snippets from online sources. In *6th IEEE/ACM International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, RAISE@ICSE 2018, Gothenburg, Sweden, May 27, 2018*, pages 21–27. ACM.

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.

Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pages 933–944. ACM.

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.

Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. 2016. A deep relevance matching model for ad-hoc retrieval. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pages 55–64. ACM.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. 2020. Momentum contrast for unsupervised visual representation learning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 9726–9735. Computer Vision Foundation / IEEE.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Junjie Huang, Duyu Tang, Linjun Shou, Ming Gong, Ke Xu, Daxin Jiang, Ming Zhou, and Nan Duan. 2021. Cosqa: 20, 000+ web queries for code search and question answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 5690–5700. Association for Computational Linguistics.

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.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Marie-Anne Lachaux, Baptiste Rozière, Marc Szafraniec, and Guillaume Lample. 2021. DOBF: A deobfuscation pre-training objective for programming languages. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 14967–14979.

Xiaonan Li, Yeyun Gong, Yelong Shen, Xipeng Qiu, Hang Zhang, Bolun Yao, Weizhen Qi, Daxin Jiang, Weizhu Chen, and Nan Duan. 2022a. Coderetriever: Large-scale contrastive pre-training for code search. In *the Association for Computational Linguistics: EMNLP 2022*.

Xiaonan Li, Daya Guo, Yeyun Gong, Yun Lin, Yelong Shen, Xipeng Qiu, Daxin Jiang, Weizhu Chen, and Nan Duan. 2022b. Soft-labeled contrastive pre-training for function-level code representation. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, Findings of ACL.

Xiang Ling, Lingfei Wu, Saizhuo Wang, Gaoning Pan, Tengfei Ma, Fangli Xu, Alex X. Liu, Chunming Wu, and Shouling Ji. 2021. Deep graph matching and searching for semantic code retrieval. *ACM Trans. Knowl. Discov. Data*, 15(5):88:1–88:21.

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*.

Sifei Luan, Di Yang, Celeste Barnaby, Koushik Sen, and Satish Chandra. 2019. Aroma: code recommendation via structural code search. *Proc. ACM Program. Lang.*, 3(OOPSLA):152:1–152:28.

Changan Niu, Chuanyi Li, Vincent Ng, Jidong Ge, Liguo Huang, and Bin Luo. 2022. Spt-code: Sequence-to-sequence pre-training for learning source code representations. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*, pages 1–13. ACM.

Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. 2021. Rocketqa: An optimized training approach to dense passage retrieval for open-domain question answering. 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 5835–5847. Association for Computational Linguistics.

Ruiyang Ren, Yingqi Qu, Jing Liu, Wayne Xin Zhao, Qiaoqiao She, Hua Wu, Haifeng Wang, and Ji-Rong Wen. 2021. Rocketqav2: A joint training method for dense passage retrieval and passage re-ranking. 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 2825–2835. Association for Computational Linguistics.

Saksham Sachdev, Hongyu Li, Sifei Luan, Seohyun Kim, Koushik Sen, and Satish Chandra. 2018. Retrieval on source code: a neural code search. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL@PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018*, pages 31–41. ACM.

Yucen Shi, Ying Yin, Zhengkui Wang, David Lo, Tao Zhang, Xin Xia, Yuhai Zhao, and Bowen Xu. 2022. How to better utilize code graphs in semantic code search? 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 722–733. ACM.

Bunyamin Sisman and Avinash C. Kak. 2013. Assisting code search with automatic query reformulation for bug localization. In *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013*, pages 309–318. IEEE Computer Society.

Luca Soldaini and Alessandro Moschitti. 2020. The cascade transformer: an application for efficient answer sentence selection. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 5697–5708. Association for Computational Linguistics.

Aäron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Yao Wan, Jingdong Shu, Yulei Sui, Guandong Xu, Zhou Zhao, Jian Wu, and Philip S. Yu. 2019. Multi-modal attention network learning for semantic source code retrieval. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*, pages 13–25. IEEE.

Xin Wang, Yasheng Wang, Yao Wan, Jiawei Wang, Pingyi Zhou, Li Li, Hao Wu, and Jin Liu. 2022. CODE-MVP: learning to represent source code from multiple views with contrastive pre-training. In *Findings of the Association for Computational Linguistics: NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pages 1066–1077. Association for Computational Linguistics.

Xin Wang, Yasheng Wang, Pingyi Zhou, Fei Mi, Meng Xiao, Yadao Wang, Li Li, Xiao Liu, Hao Wu, Jin Liu, and Xin Jiang. 2021. CLSEBERT: contrastive learning for syntax enhanced code pre-trained model. *CoRR*, abs/2108.04556.

Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. 2017. End-to-end neural ad-hoc ranking with kernel pooling. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, pages 55–64. ACM.

Ziyu Yao, Daniel S. Weld, Wei-Peng Chen, and Huan Sun. 2018. Staqc: A systematically mined question-code dataset from stack overflow. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 1693–1703. ACM.

Hang Zhang, Yeyun Gong, Yelong Shen, Jiancheng Lv, Nan Duan, and Weizhu Chen. 2022. Adversarial retriever-ranker for dense text retrieval. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.