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

Hande Dong
International Digital Economy Academy
Shenzhen, China
donghd66@gmail.com

Jiayi Lin
International Digital Economy Academy
Shenzhen, China
linjiayi@idea.edu.cn

Yichong Leng
University of Science and Technology of China
Hefei, China
lyc123go@mail.ustc.edu.cn

Jiawei Chen
Zhejiang University
Hangzhou, China
sleepyhunt@zju.edu.cn

Yutao Xie
International Digital Economy Academy
Shenzhen, China
yutaoxie@idea.edu.cn

Abstract—Pretrained Language Models (PLMs) have emerged as the state-of-the-art paradigm for code search tasks. The paradigm involves pretraining the model on search-irrelevant tasks such as masked language modeling, followed by the finetuning stage, which focuses on the search-relevant task. The typical finetuning method is to employ a dual-encoder architecture to encode semantic embeddings of query and code separately, and then calculate their similarity based on the embeddings.

However, the typical dual-encoder architecture falls short in modeling token-level interactions between query and code, which limits the model's capabilities. In this paper, we propose a novel approach to address this limitation, introducing a cross-encoder architecture for code search that jointly encodes the semantic matching of query and code. We further introduce a Retriever-Ranker (RR) framework that cascades the dual-encoder and cross-encoder to promote the efficiency of evaluation and online serving. Moreover, we present a probabilistic hard negative sampling method to improve the cross-encoder's ability to distinguish hard negative codes, which further enhances the cascade RR framework. Experiments on four datasets using three code PLMs demonstrate the superiority of our proposed method.
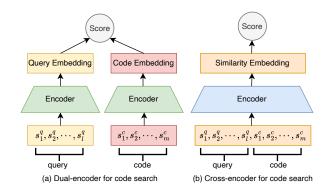
Fig. 1. The dual-encoder architecture and the cross-encoder architecture for code search. $s_1^q, s_2^q, \cdots, s_l^q$ denotes the token sequence of query q, and $s_1^c, s_2^c, \cdots, s_m^c$ denotes the token sequence of code c. (a) In dual-encoder, we input the token sequence of the query and code separately; (b) In cross-encoder, we input the token sequence concatenation of the query and the code.

I. INTRODUCTION

Code search is a crucial task in software engineering, which aims to retrieve relevant code snippets from large code repositories for a given natural language query [1]–[3]. 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 [4]. Specialized rules are often designed to extract features of codes [5], and term frequencies are exploited to identify relevant queries and codes [6]. However, traditional IR methods have some inherent limitations; for instance, the designed rules are hard to cover complex and implicit features about the matching between queries and codes and the term mismatch issue can limit the ability to discover useful codes.

Recently, deep learning techniques, particularly Pretrained Language Models (PLMs), have demonstrated remarkable success in the code search task, thanks to their ability to learn complex and implicit features [1], [7]. The PLMs employed in code search tasks generally undergo two stages: pretraining and finetuning. Pretraining involves training the model with universal pretext tasks to enhance its code understanding ability. These tasks include masked language modeling [8], identifier prediction [7], contrastive learning [9], editing [10], and more. Finetuning involves training the model with the search task to improve its code search ability. Specifically, the search task aims to maximize the relevance score between relevant query and code while minimizing the relevance score between irrelevant query and code. While previous research on PLMs for code search primarily focuses on the pretraining stage, i.e., proposing various training tasks to enhance the model's code understanding ability, there has been little focus on the finetuning stage. Thus, our paper mainly focuses on improving the finetuning stage for code search.

During the fine-tune stage, pre-trained language models (PLMs) typically employ a dual-encoder architecture for code search, as shown in Figure 1(a). This architecture separately encodes the query and code to obtain their embeddings as

semantic representations and then computes the relevance score between them based on these embeddings [9]–[11]. However, while the dual-encoder can fully model the token-level interactions within the query and code separately using self-attention mechanisms [12], [13], it cannot model the token-level interactions between them. As a result, the dual-encoder's capability to learn fine-grained interactions between query and code tokens is limited.

In this paper, we propose Retriever and Ranker with Probabilistic Hard Negative Sampling method (R2PS) for code search, which is a patch designed to be used with different code PLMs. Compared to previous work that finetunes code PLMs with the dual-encoder architecture, R2PS has the following three advantages:

a) Strong Model Capability: To overcome the dual-encoder's limitations in model capability, we propose a cross-encoder architecture for code search, as illustrated in Figure 1(b). The cross-encoder first concatenates the query and code tokens and then encodes the concatenated query and code together. This approach allows the self-attention mechanism in the PLM encoder to model the token-level interactions between query and code, thereby improving the model's capability.

b) Balance of Effectiveness and Efficiency: While effective, the cross-encoder requires concatenating each query with all codes in the codebase and encoding all the concatenations for evaluation and online serving. This approach is infeasible when dealing with large codebases containing millions of code snippets. On the other hand, the dual-encoder architecture can encode all codes in the codebase during data pre-processing and only needs to encode the query to find relevant codes based on their embeddings during online serving. To take advantage of both the cross-encoder's effectiveness and the dual-encoder's efficiency, we introduce a retriever-ranker framework for code search. This framework consists of a dual-encoder as the retriever module to retrieve a small set of possibly relevant codes for the query and a cross-encoder as the ranker module to further rank this small set of codes. In this way, the cross-encoder only needs to encode the query and the retrieved codes, making it efficient for evaluation and online serving.

c) Reasonable Training Method: To further improve the performance of the cascade RR framework, we propose a probabilistic hard negative sampling strategy to sample negative samples to train the cross-encoder. We remove two kinds of codes from the candidate negative samples according to the relevance scores of the queries and codes calculated by the well-trained dual-encoder. First, we remove codes with low relevance scores, indicating that these codes are irrelevant to the queries. This approach ensures that the cross-encoder only samples codes with high relevance scores as hard negative samples, improving its ability to distinguish relevant code from codes with high relevance scores retrieved by the dual-encoder. Second, we remove codes with very high relevance scores, indicating that these codes are possibly relevant to the queries but are not labeled in the dataset. This approach mit-igates the risk of sampling false negative samples (unlabeled positive samples) to train the model, thereby improving its performance. Finally, we rescale the relevance scores of the remaining codes returned by the dual-encoder as the sample probability and sample codes according to this probability as negative samples to train the cross-encoder.

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

- We identify the limitations of the dual-encoder architecture and propose an alternative approach using the cross-encoder architecture to improve model's capability for code search.
- We introduce the RR framework, which leverages the strengths of both cross-encoder and dual-encoder architectures for code search.
- We propose a novel sampling strategy, probabilistic hard negative sampling, which more effectively trains the cross-encoder model by sampling negative codes.
- We Conduct comprehensive experiments on four datasets using three code encoders, and demonstrate the superiority of our proposed method.

II. RELATED WORK

In this section, we briefly review the related work from three aspects, including code search, pretrained language models for software engineering, and finetuning PLMs for code search.

A. Code Search

Code search aims to find relevant codes for given queries [14], [15]. The core of code search is to predict whether queries are relevant to codes [1], [16]. To determine the relevance score, traditional information retrieval (IR) methods mainly 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 [4]–[6]. 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.

To overcome the limitations of IR-based methods, deep learning has been applied to the code search task to learn semantic representations of queries and codes [15]. The relevance score is typically calculated using a factorized query embedding and code embedding [2]. The query embedding is generated using natural language processing (NLP) models such as fastText [15], LSTM [1], or GRU [2]. Various methods have been employed to calculate the code embedding, including LSTM [17], MLP [1], and Graph Neural Network [3]. Code can be represented by various data structures through code static analysis [5], such as Abstract Syntax Tree (AST) [3], variable and function name [1]. To encode these different structures, a variety of models have been proposed [17]. While these techniques reduce the need for human-designed rules, they still require human effort to create effective data structures and train the models on them.

B. Pretrained Language Models (PLMs)

a) PLMs for NLP: Recent years have seen significant success in the field of natural language processing (NLP) thanks to pre-trained language models (PLMs), which have become the primary paradigm in this area [18]. These models are transformer-based and trained with self-supervised learning using a large set of unlabeled data. The most commonly used pre-training task involves the "mask then predict" strategy, where tokens in a sequence are masked and predicted based on the surrounding tokens. For instance, BERT [18] masks tokens randomly and predicts them based on the left tokens, while GPT [19] masks the latter tokens in a sequence and predicts them based on former tokens. Pre-trained models can then be fine-tuned on many downstream tasks such as sentiment classification and summarization to transfer PLM knowledge, leading to state-of-the-art performance in these tasks.

b) PLMs for Software Engineering: Motivated by the huge succees of pretrained language models (PLMs) in the NLP field, many researchers introduce PLMs to software engineering field [10], [11], [20]–[22]. Various code PLMs have been trained from different perspectives, with some based on language modeling [8], [23], and others based on code-relevant tasks such as link prediction on abstract syntax tree (AST) and data flow graph (DFG) [7], [11], or contrastive learning with regard to multi-modals about code [9]. Following pretraining, code PLMs can be finetuned on code downstream tasks, such as code search and code generation, significantly outperforming previous models.

While much of the research on code PLMs focuses on designing different pretraining tasks to improve their general ability, little attention has been given to improving the fine-tuning stage for specific downstream tasks. This suggests that the potential of PLMs for downstream tasks has not been fully exploited. In this paper, we focus on improving the finetuning stage of PLMs for the code search task, exploring ways to make the models more suitable for this task.

C. Finetuning PLMs for Code Search

When using pre-trained language models (PLMs) for code search, the most commonly employed approach is the dual-encoder architecture [7], [9], [21]. In this method, the PLM serves as an encoder to extract query and code embeddings, which are then used to compute the relevance score. Notably, the dual-encoder architecture for fine-tuning code PLMs is similar to prior deep learning approaches for training models on code search tasks [1]. The key difference lies in the choice of encoder. While previous work utilized encoders such as LSTM and GNN, the PLM based models relies on the pretrained Transformer encoder.

To optimize the model, various loss functions have been adopted for the code search task. One approach is to treat the task as a binary classification problem, where the objective is to determine whether a query is relevant to a code or not. In this case, a binary classification loss can be utilized to optimize the code search model, as proposed in [24]. However, the code search task requires ranking all codes according to their relevance scores with a query, is actually a ranking task. Therefore, the binary classification loss is not entirely consistent with the ranking setting. To address this, some previous works have proposed using the pairwise loss to optimize the code search model. This approach involves maximizing the relevance score margin between a query and its relevant code compared to its irrelevant code [1], [3], [17], [25]. Furthermore, other studies have proposed optimizing the model by maximizing the relevance scores of a query with its relevant code and simultaneously minimizing the relevance scores of this query with many irrelevant codes [7], [20], [26].

III. METHOD

In this section, we present a retriever and ranker framework with a probabilistic hard negative sampling method for code search, which is a flexible and universal patch and can be applied to many code PLMs to improve the performance of previous work for the code search task. We begin by introducing the code search task and the dual-encoder architecture which is typically used for finetune PLMs for code search. We then introduce the cross-encoder architecture and the retrieval and ranker framework for the code search task. We then delve into the details of the probabilistic hard negative sampling method, which is used to train the cross-encoder within the RR framework. Finally, we provide an in-depth analysis of RR/R2PS, highlighting its superior model capabilities and complexity.

A. Task Formulation and Dual-encoder

Assuming we have a large 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 quickly identify and present a small amount of relevant codes to the user based on relevance scores (also known as similarities). The core of code search lies in two perspectives: (1) Precision: precisely estimate the relevance scores of queries and codes; (2) Efficiency: rapidly estimate the relevance scores of the query and all codes in the code corpus. The definitions for used notations in this paper are shown in Table I.

As depicted in Figure 1(a), the dual-encoder architecture is commonly used to predict relevance scores for PLM-based code search. Firstly, the query and code are tokenized to token sequence. Then, the PLM encoder maps the query and code token sequence to query embedding and code embedding, respectively, Finally, the relevance score of the query and code is calculate by the dot product of the two embeddings. The dual-encoder architecture can be formulated as:

$$s_{dual}(q, c) = < E(q), E(c) >, \tag{1}$$

where $E()$ denotes the PLM encoder, $<, >$ denotes the dot product operation, q and c denote the token sequence of query and code, respectively. The PLM is essentially a transformer-based model, with self-attention of all tokens in the sequence to learn interactions among the tokens [27].

TABLE I
NOTATION AND DEFINITIONS.

Notation	Annotation
q_i, c_i	A query, a code snippet. In some context, they denote the token sequence of the tokenized query and code.
$s_1^q, s_2^q, \cdots, s_l^q$	The query token sequence of query q; l is the token sequence length.
$s_1^c, s_2^c, \cdots, s_m^c$	The query token sequence of query c; m is the code sequence length.
$E()$	The code pretrained language model, which is Transformer-based model.
$<,>$	Dot product of two vectors.
$\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$	The query, key, value matrix in Transformer, $\boldsymbol{Q}$ is irrelevant with user query q.
$\boldsymbol{A}$	The attentin matrix in Transformer, with $\boldsymbol{A} = softmax(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d}})$.
$\boldsymbol{H}$	The representation matrix of token sequence.
$[q, c]$	The concatenation of query token sequence and code token sequence.
$s(q, c)$	The relevance score of the query q and the code c.

B. Cross-encoder and RR Framework

a) Motivation: Trasnformer-based models [28] are powerful thanks to their self-attention mechanism that enables them to model full token interactions within a sequence. In the Transformer architecture, self-attention can be formulated as follows:

$$\boldsymbol{H}' = softmax(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d}})\boldsymbol{V}, \tag{2}$$

where $\boldsymbol{Q} = \boldsymbol{H}\boldsymbol{W}_1$, $\boldsymbol{K} = \boldsymbol{H}\boldsymbol{W}_2$, $\boldsymbol{V} = \boldsymbol{H}\boldsymbol{W}_3$ are linearly projected by the hidden representation $\boldsymbol{H}$, $\boldsymbol{H}$ denotes the representation matrix of all tokens in the sequence, with each row corresponding to a specific token, $\boldsymbol{H}'$ denotes the representation matrix after self-attention operation, $\sqrt{d}$ denotes the scaling factor, $softmax(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d}})$ is attention matrix of Transformer.

From element perspective, the i-th token in the sequence can be formualted as:

$$\boldsymbol{h}_i' = \sum_{j=1}^{l} st(<\boldsymbol{q}_i, \boldsymbol{k}_j>)\boldsymbol{v}_j, \tag{3}$$

where $\boldsymbol{h}_i'$, $\boldsymbol{q}_i$, $\boldsymbol{k}_j$, $\boldsymbol{v}_j$ denote the representation vectors of i/j-th token in the corresponding matrix, $<\boldsymbol{q}_i, \boldsymbol{k}_j>$ denotes dot-product of vector $\boldsymbol{q}_i$ and $\boldsymbol{k}_j$ which can be regarded as the interaction of i-th and j-th tokens, $st()$ denotes the scaling operation and softmax operation about $j = 1, 2, \cdots, l$. Thus, the embedding of each token, $\boldsymbol{h}_i'$, is obtained by modeling its interactions with all tokens in the sequence ($<\boldsymbol{q}_i, \boldsymbol{k}_j>, j = 1, 2, \cdots, l$) and fusing all tokens information according to the interactions.

Therefore, the dual-encoder for code search can model the token-level interactions within both query tokens and code tokens with Transformer-based encoder $E()$. Specifically, $E(q)$ models all interactions in the query token sequence $q = (s_1^q, s_2^q, \cdots, s_l^q)$, including all pairs (s_i^q, s_j^q), where $i, j = 1, 2, \cdots, l$; Similarly, $E(c)$ models all interactions in the code token sequence $c = (s_1^c, s_2^c, \cdots, s_m^c)$, including all pairs (s_i^c, s_j^c), where $i, j = 1, 2, \cdots, m$. However, the model cannot directly model token-level interactions between query and code tokens (s_i^q, s_j^c) where $i = 1, 2, \cdots, l$ and $j = 1, 2, \cdots, m$, This limitation restricts the model's capability and effectiveness in code search.

b) Cross-encoder: To improve the model capability of PLMs for code search, we propose using a cross-encoder to model the token-level interactions between query and code. Specifically, we concatenate the query token sequence and code token sequence to a unified sequence, then use the PLM encoder to map the unified sequence into an embedding, and finally use a neural network head to map the embedding to a scalar that represents the relevance score of the query and code. The cross-encoder architecture can be represented as:

$$s_{cross}(q, c) = NN(E([q, c])), \tag{4}$$

where $[q, c] = (s_1^q, s_2^q, \cdots, s_l^q, s_1^c, s_2^c, \cdots, s_m^c)$ denotes the concatenation of query and code, $E()$ denotes the PLM encoder, and $NN()$ denotes the neural network head.

In this way, the query tokens and code tokens are inputed into the Transformer model together, and thus all token-level interactions can be effectively modeled, including not only interactions within query (s_i^q, s_j^q) and code (s_i^c, s_j^c), but also those between query and code $(s_i^q, s_j^c), (s_j^c, s_i^q)$. As a result of its ability to learn the cross-interactions between query and code, we refer to this model as the cross-encoder framework. Compared to the dual-encoder, the model capability of the cross-encoder is stronger, making it more precise for code search.

c) Retriever and Ranker Framework: During the inference stage, the cross-encoder must encode all possible concatenations of each query and all codes in the codebase using the PLM encoder. This enables the cross-encoder to retrieve the relevant code snippets based on their respective relevance scores. However, given that the codebase is typically vast and comprises millions of codes, this process demands a significant amount of computing resources. As a result, the cross-encoder can be slow to provide results when serving online, which can impact its efficiency.

To make the cross-encoder practical for code search, we introduce a Retriever and Ranker framework (RR) for code search, as illustrated in Figure 2. The RR framework comprises two cascaded modules: (1) A dual-encoder module is employed as the retriever to identify k codes with the highest relevance scores for a given user query; (2) A cross-encoder module is used as the ranker to rank the k codes further.

We explain the rationality behind the efficiency of the RR framework for code search as follows: In the dual-encoder model, we can compute the code embeddings of all the codes in the codebase during pre-processing and store them as $E(c_i)$, where $i = 1, 2, \cdots, N$. Therefore, during evaluation and online serving, the retriever only needs to compute the embedding of the given query, $E(q)$, and match it with the pre-calculated code embeddings to calculate the relevance scores. The cross-encoder model then determines the relevance scores between the query and the retrieved k codes, and ranks them accordingly. As a result, the online encoding time for
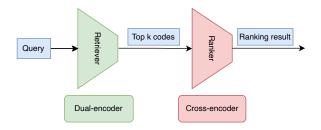
Fig. 2. Our proposed Retriever and Ranker framework for code search.

Fig. 3. Our proposed probabilistic hard negative sampling method.

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. This computational process 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, which results in better performance. However, with a smaller value of k, the cross-encoder performs fewer propagations, which reduces the computing cost. Thus, choosing the appropriate value of k can balance the performance and computing cost.

C. Training Method

The cascaded RR framework consists of two modules that perform distinct functions. The dual-encoder module retrieves data from the codebase, which contains all codes, and its ability to predict the relevance scores of all codes is crucial. On the other hand, the cross-encoder module ranks a small number of retrieved codes that have high relevance scores, as determined by the dual-encoder. Thus, its ability to predict the relevance scores of these potentially relevant codes is of utmost importance. Therefore, it is essential to design different training goals for the two modules to optimize their performance.

The goal of training a code search model is to learn a relevance estimation function that assigns higher scores to relevant codes for queries compared to those that are irrelevant. To achieve this, we sample some irrelevant codes as negative samples and aim to maximize the relevance scores of the relevant codes while minimizing the relevance scores of the sampled codes. Our method employs the InfoNCE loss [29], [30] as loss function, which can be formulated as:

$$\mathcal{L}_q = -log \frac{e^{s(q,c^+)/\tau}}{e^{s(q,c^+)/\tau} + \sum_{i=1}^{m} e^{s(q,c_i^-)/\tau}}, \quad (5)$$

where q is a query, c^+ is a relevant code of the query q, and c_i^- is the set of sampled negative codes, τ denotes the temperature hyper-parameter, $s(q,c)$ denotes the relevance score of the query-code pair estimated by the PLM encoders. Minimizing this loss will result in increasing the relevance score of the relevant query-code pair $s(q,c^+)$ and decreasing the relevance scores of the query with irrelevant codes $s(q,c_i^-)$.

As mentioned before, it is crucial for the dual-encoder to predict the relevance scores of queries with all codes in the codebase. Therefore, our primary objective is to train the model to enhance its ability t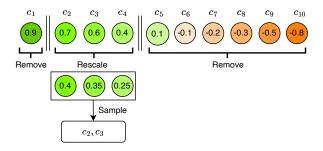o retrieve potentially relevant codes from the entire codebase. To achieve this, we utilize a method of negative sampling wherein we randomly select codes from the entire codebase as negative samples to train the model. In practice, we employ the in-batch negative sampling technique, which treats the codes in the sampled batch as negative samples. Specifically, given a batch data $\{(q_j, c_j)\}_{j=1}^{b}$, where code c_j is relevant to query q_j, all other codes c_i, where $i \neq j$ are deemed as negative samples of query q_j. Since the batch data is randomly sampled from the training set, we can consider other codes in the batch as randomly sampled from the training set.

On the other side, it is crucial for the cross-encoder to predict the relevance scores the query with possibly relevant codes retrieved by the dual-encoder. To achieve this, we should train the model to improve its ability to identify relevant codes from those retrieved by the dual-encoder. In pursuit of this objective, we utilize a set of codes predicted to be possibly relevant by the dual-encoder as negative samples for the cross-encoder. The dual-encoder is capable of predicting the relevance scores of each query to all codes, and we can select the codes with high relevance scores as the potentially relevant ones. However, we must exercise caution, as the small set of codes with the highest relevance scores may contain false negatives, which are relevant but not labeled. To prevent the inclusion of such false negatives, we exclude the codes with the highest relevance scores predicted by the dual-encoder from our sampling process.

To implement above idea, we propose a probabilistic hard negative sampling method, as shown in Figure 3. The specific process contains the following steps. We first calculate the relevance scores of the query q with all codes c_i in the training set using the well-trained dual-encoder. We then eliminate two types of codes from the sampling candidates: (1) the (probably) irrelevant codes whose relevance scores are below a threshold ρ_1, and (2) the (probably) false negative codes whose relevance scores are above a threshold ρ_2. Since the number of irrelevant codes is much greater than the number of false negative codes, the remaining codes are still relatively hard negative samples. We refer to these remaining codes as $\{c_i^*\}_{i=1}^{L}$, which satisfy the condition $\rho_1 < s_{dual}(q,c_i^*) < \rho_2$. Next, we rescale the relevance scores of left code candidates

as a probability distribution using the following formula:

$$p_i = \frac{e^{s_{dual}(q,c_i^*)/\tau'}}{\sum_{j=1}^{L} e^{s_{dual}(q,c_j^*)/\tau'}}, i = 1, 2, \cdots, L, \qquad (6)$$

and then sample m negative samples according to this probability distribution as negative samples. The parameter τ' adjusts the smoothness of the distribution; larger τ' results in a smoother distribution, allowing us to sample more codes with higher relevance scores according to the dual-encoder. After sampling m negative codes, we can calculate the InfoNCE loss function and optimize the cross-encoder model.

Through the use of a probabilistic hard negative sampling strategy, the cross-encoder is able to be more effectively trained to serve as the ranker module within the cascaded RR framework. This framework is referred to as R2PS. Additionally, the cross-encoder can be optimized by implementing an in-batch negative sampling method as the dual-encoder, which we refer to as RR in our paper.

D. Model Analysis

We conduct some analysis to exhibit the superiority of the RR/R2PS method. First, we demonstrate the model capability of the cross-encoder and provide rationalization for its effectiveness in comparison to the dual-encoder. We then present a complexity analysis of the dual-encoder, cross-encoder, and RR framework to demonstrate the efficiency of the RR framework for evaluation and online serving. Finally, we discuss the process of fine-tuning CodeBERT, which is the work most closely related to our paper.

a) Model Capability: The self-attention mechanism for the query tokens in the dual-encoder can be formualted as:

$$\boldsymbol{H}^q = \boldsymbol{A}^{qq}\boldsymbol{V}^c, \qquad (7)$$

where $\boldsymbol{A}^{qq} \in \mathbb{R}^{l*l}$ denotes the attention matrix to model the interactions between all query tokens, $\boldsymbol{V}^q, \boldsymbol{H}^q \in \mathbb{R}^{l*d}$ is the representation matrix of all query tokens. Similarly, the self-attention mechanism for the code tokens in the dual-encoder can be formualted as:

$$\boldsymbol{H}^c = \boldsymbol{A}^{cc}\boldsymbol{V}^c, \qquad (8)$$

where $\boldsymbol{A}^{cc} \in \mathbb{R}^{m*m}$ denotes the attention matrix to model the interactions between all token tokens, $\boldsymbol{V}^c, \boldsymbol{H}^c \in \mathbb{R}^{m*d}$ is the representation matrix of all code tokens. These two equations can be combined into one as follows:

$$\begin{pmatrix} \boldsymbol{H}^q \\ \boldsymbol{H}^c \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}^{qq} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}^{cc} \end{pmatrix} \begin{pmatrix} \boldsymbol{V}^q \\ \boldsymbol{V}^c \end{pmatrix}. \qquad (9)$$

The self-attention mechanism of all tokens in the cross-encoder can be formualted as:

$$\begin{pmatrix} \boldsymbol{H}^q \\ \boldsymbol{H}^c \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}^{qq} & \boldsymbol{A}^{qc} \\ \boldsymbol{A}^{cq} & \boldsymbol{A}^{cc} \end{pmatrix} \begin{pmatrix} \boldsymbol{V}^q \\ \boldsymbol{V}^c \end{pmatrix}, \qquad (10)$$

where $\boldsymbol{A}^{qc} \in \mathbb{R}^{l*m}, \boldsymbol{A}^{cq} \in \mathbb{R}^{m*l}$ denote the interactions between query and code tokens.

Comparing Equation (9) and Equation (10), we can find that the dual-encoder is actually a special case of the cross-encoder


```
Query: save the pytorch model to pickle
located at path

Code:
def save_model(path):
    file_name = os.path.join(path, 'model.bin')
    model_state = torch.load(file_name)
    saved_file_name = os.path.join(path, 'model_state.pkl')
    pickle.dump(model_state, open(saved_file_name, 'wb'))
```

Fig. 4. A case of a query and its relevant code. The same colors indicate that these tokens match between query and code.

with $\boldsymbol{A}^{qc} = \boldsymbol{0}$ and $\boldsymbol{A}^{cq} = \boldsymbol{0}$. Hence, it is apparent that the cross-encoder possesses a stronger model capability than the dual-encoder.

In Figure 4, we demonstrate the importance of query-code interactions modeled by $\boldsymbol{A}^{qc}$ and $\boldsymbol{A}^{cq}$ using a case study. To accurately predict the relevance of a query and code, it is essential to examine the various matching relationships between them in detail, as depicted by the same colors in the figure. The matching query and code tokens facilitate strong interactions in $\boldsymbol{A}^{qc}$ and $\boldsymbol{A}^{cq}$, which result in that their representations are enhanced by fusing information within these matching tokens each other with $\boldsymbol{A}^{qc}$ and $\boldsymbol{A}^{cq}$. This approach enables the model to learn the matching relationships between query and code tokens more effectively.

b) Complexity Analysis: Assuming that we have M queries to provide them with relevant codes from a codebase containing N codes, we analyze the complexity about the evaluation with different framework. The search process entails computing the relevance scores of all queries against all codes. These relevance scores are represented by a matrix $\boldsymbol{S} \in \mathbb{R}^{M*N}$, where each row corresponds to a query and each column represents a code.

With the dual-encoder architecture, $\boldsymbol{S}$ can be factorized by multiplying the embedding matrix of all queries and codes, as expressed in the following equation:

$$\boldsymbol{S} = \boldsymbol{Q}\boldsymbol{C}^T, \qquad (11)$$

where $\boldsymbol{Q} \in \mathbb{R}^{M*d}$ and $\boldsymbol{C} \in \mathbb{R}^{N*d}$, d denotes the embedding dimension. Each row in $\boldsymbol{Q}$ and $\boldsymbol{C}$ is the output of PLM encoder for corresponding query and code. Therefore, M rounds of PLM propagation for queries and N rounds of PLM propagation for codes are sufficient. As a result, the dual-encoder approach offers a complexity of $\mathcal{O}(M + N)$.

On the contrary, the dual-encoder cannot factorize $\boldsymbol{S}$ as the dual-encoder. Each element in $\boldsymbol{S}$ must be calculated by propagation of the PLM encoder. $\boldsymbol{S}$ consists of $M * N$ elements, making the complexity of the cross-encoder $\mathcal{O}(M * N)$.

The RR framework comprises of two cascade modules. Firstly, a dual-encoder is used to retrieve relevant codes from the entire codebase, which has a complexity of $\mathcal{O}(M + N)$. Secondly, a cross-encoder is employed to identify relevant codes from the k retrieved codes. For every query, the cross-encoder needs to conduct forward propagation k times, re-

TABLE II
COMPLEXITY OF DIFFERENT CODE SEARCH FRAMEWORKS.

Framework	Dual-encoder	Cross-encoder	RR framework
Complexity	$\mathcal{O}(M + N)$	$\mathcal{O}(M * N)$	$\mathcal{O}(M * (1 + k) + N)$

TABLE III
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

sulting in a complexity of $\mathcal{O}(M * k)$ for the cross-encoder in the RR framework. Therefore, the total complexity of the RR framework can be expressed as $\mathcal{O}(M * (1 + k) + N)$.

Table II summarizes the complexities of the three frameworks discussed above. The dual-encoder's complexity of $\mathcal{O}(M + N)$ makes it highly efficient to implement for evaluation. However, the cross-encoder's complexity of $\mathcal{O}(M * N)$ makes it infeasible to implement, as the number of queries and codes are often immense. In contrast, the complexity of the RR framework, given that k is always a small number, is of the same magnitude as that of the dual-encoder but much lower than that of the cross-encoder. Therefore, the RR framework is also highly efficient to implement.

c) Relation with Finetuning CodeBERT: CodeBERT [8] was the pioneering work that leveraged PLM for the code search task. It utilizes the cross-encoder framework for finetuning and trains the search model using binary classification loss, which suffers from the efficiency problem and is not practical for evaluation purposes on all codes in the codebase. The official code of CodeBERT to evaluate their model by ranking the codes from a batch of 1000 codes rather than the full codebase, which is inconsistent with real scenario and may lead to inaccuracies due to high variance.

To address the issue of inconsistency between evaluation and real-world application, the research team behind CodeBERT adopted a different approach in their subsequent work [7], [9]. They abandoned the previous method and instead used a dual-encoder framework for code search. Furthermore, based on the experimental results reported in their later work, it appears that they also re-implemented CodeBERT using the dual-encoder framework and reported the results of finetuning CodeBERT with the dual-encoder framework in their subsequent work. In this paper, we have followed their approach and re-implemented the finetune settings for CodeBERT accordingly.

Even compared to the original CodeBERT implementation which uses a cross-encoder in the official code [8], our method differs in three key ways. First, we use the RR framework to efficiently cascade the dual-encoder and cross-encoder components. Second, we train our model using rank loss and leverage a probabilistic hard negative method to enhance the training of the cross-encoder. This rank loss differs from the classification loss used in the original CodeBERT implementation and is better suited for a code search setting. Finally, we evaluate our model using all codes in the codebase as candidate codes, whereas CodeBERT only considers a batch of 1000 codes for evaluation.

IV. EXPERIMENTS

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

- **RQ1:** Can our proposed RR/R2PS patch boost the performance of PLMs for the code search task?
- **RQ2:** Is the cross-encoder inefficient for online serving? If so, can our RR framework overcome this efficiency issue?
- **RQ3:** How does the number of retrieved code snippets impact the performance and efficiency tradeoff?
- **RQ4:** Does our R2PS method provide a more reasonable ranking distribution of relevant code snippets?

A. Experimental Setup

Our proposed RR/R2PS is actually a patch during the finetuning stage for the code search task, which can be applied to various pretrained language models designed for code. To fully validate the effectiveness and universality of our proposed method, we use three different code PLMs as backbone models of our RR/R2PS patch, including CodeBERT [8], GraphCodeBERT [7], and UniXcoder [9].

a) Datasets: We evaluate using four code search benchmark datasets: CodeSearchNet (CSN) [7], [31], AdvTest [32], StaQC [33], and CosQA [24]. 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 finetuend CodeBERT encoder. Table III summarizes the dataset statistics, including the number of relevant 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.

b) 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 [9].

MRR calculates the average reciprocal rank of the relevant codes for all queries. It is defined as:

$$MRR = \frac{1}{|D|} \sum_{(q,c) \, in \, D} \frac{1}{rank_c^{(q)}},$$ (12)

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

c) Baselines: The compared methods include: (1) CodeBERT [8], which was pretrained on the masked language modeling and replaced token detection task; (2) GraphCodeBERT [7], which was pretrained on additional graph relevant tasks such as link prediction of data flow graph; (3) UniXcoder [9], which was pretrained on addtional language modeling task; (4) SyncoBERT [34], which was pretrained on syntax-enhanced contrastive learning task; (5) CodeRetriever [20], which was pretrained on NL-code contrastive learning task. The reported results of CodeBERT, GraphCodeBERT, and UniXcoder are reproduced by us by re-run their official released code, and the performance is nearly the same as their reported results in their paper.

d) Implementation Details: During the finetune stage, both the query encoder and the code encoder in the dual-encoder share parameters, with the same code pre-trained language model (PLM) used to initialize the two encoders. The cross-encoder is also finetuned based on the corresponding code PLMs. However, it does not share parameters with the previous query and code encoders in the dual-encoder framework. In the RR approach, we train the dual-encoder and the cross-encoder together using in-batch negative sampling strategy. On the other hand, the R2PS approach involves a two-step training process: first, we train the dual-encoder with in-batch negative sampling; then, we train the cross-encoder with negative sampling based on relevance scores calculated by the previous well-trained dual-encoder.

We follow the same experimental settings as the series work of CodeBERT, GraphCodeBERT, and UniXcoder, 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.

B. Performance Comparison(RQ1)

The performance of compared methods in terms of MRR is shown in Table IV. The RR patch was applied to CodeBERT, GraphCodeBERT, and UniXcoder's backbone models, while the R2PS patch was applied only to UniXcoder, due to computing resource limitation. From this table, we observe the following: (1) Our proposed RR patch significantly improves the performance of all three backbone models, with an average improvement of 4.7 on CodeBERT, 4.1 on GraphCodeBERT, and 2.8 on UniXcoder across the four datasets. Furthermore, the R2PS patch leads to a substantial improvement of 4.1 MRR for UniXcoder. These results demonstrate the effectiveness of our proposed RR/R2PS method for code search. (2) The performance enhancement of UniXcoder with the RR patch is more significant than GraphCodeBERT and CodeBERT, consistent with their performance without the RR patch. This finding suggests that better-performing code PLMs can achieve even stronger performance when used in conjunction with the RR patch. (3) Overall, UniXcoder with R2PS outperforms all other methods, including the latest CodeRetriever, providing further evidence for the superiority of our proposed method.

C. Efficiency Evaluation (RQ2)

We conducted a comparison of the computing costs of the dual-encoder, cross-encoder, and our RR framework during the online serving stage. To simulate incoming queries in online serving, we simulate the response processing time when the server receives a user query and infers relevant codes before returning them to the user. The code embeddings were pre-calculated in the pre-processing stage, and the response time did not include the time required to embed codes for the dual-encoder. We conducted experiments with codebase scales of 1,000, 10,000, and 100,000 codes, and calculated the average response time for 100 query requests, using an A100 GPU. Our results, presented in Table V, show that: (1) The dual-encoder and the RR achieve response times of no more than 100ms for a codebase with 100,000 codes, which is efficient for online serving. In contrast, the cross-encoder takes approximately 8 minutes to respond for a codebase of 100,000 codes, which is too long for online serving. (2) The response time of the RR is longer than that of the dual-encoder due to the ranker module in the RR framework. However, the extra time is justified considering the significant performance improvement. (3) As the codebase scale increased by 10 times, the computing time for the cross-encoder increases by about 10 times, while the computing time for the dual-encoder and RR increases slowly (much less than 10 times). This is because the cross-encoder requires the encoding of the concatenation of the given queries and all codes in the codebase during online serving, while the dual-encoder and RR framework calculate the embeddings of all codes in the codebase during the data pre-processing stage.

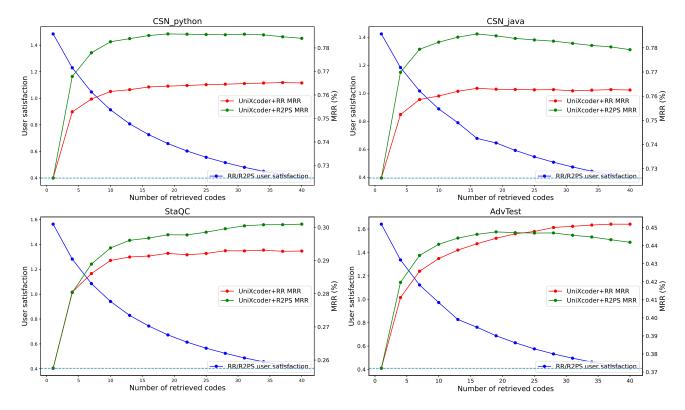| Method	Ruby	JS	Go	Python	Java	PHP	Average	CosQA	StaQC	AdvTest
SyncoBERT	72.2	67.7	91.3	72.4	72.3	67.8	74.0	-	-	38.1
CodeRetriever	75.3	69.5	91.6	73.3	74.0	68.2	75.3	69.6	-	43.0
CodeBERT	68.8	62.7	89.0	68.5	68.4	64.2	70.3	66.9	21.4	34.9
+RR	75.9	68.9	91.8	74.9	74.5	69.4	75.9(↑5.6)	68.3(↑1.4)	26.2(↑4.8)	42.0 (↑7.1)
GraphCodeBERT	69.4	65.4	90.3	70.8	70.0	64.1	71.7	68.7	20.5	39.1
+RR	75.8	71.8	92.5	73.6	77.1	71.0	77.0(↑5.3)	69.8(↑1.1)	24.3(↑3.8)	45.4(↑6.3)
UniXcoder	73.9	68.5	91.5	72.5	72.6	67.6	74.4	69.6	25.8	42.5
+RR	77.9	72.1	92.5	76.1	76.0	70.6	77.5(↑3.1)	71.5(↑1.9)	29.0(↑3.2)	45.4(↑2.9)
+R2PS	**79.1**	**73.8**	**93.4**	**78.3**	**78.2**	**72.4**	**79.2**(↑4.8)	**72.3**(↑2.7)	**29.4**(↑3.6)	**47.0**(↑5.2)

Fig. 5. Performance and user satisfaction of the RR and R2PS patched UniXcoder with the different number of retrieved code k. The below dashed line is the performance of UniXcoder without RR or R2PS patch.

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

D. Hyper-parameter Analysis (RQ3)

We conducted experiments to investigate the impact of varying the number of retrieved codes, k, on both the performance and user satisfaction of our framework. Specifically, we measured user satisfaction by evaluating their tolerance for the response time of the system. To this end, we defined the user satisfaction score, S, as follows:

$$S = \frac{100}{t + 50},\qquad(13)$$

where t denotes the response time of the model in milliseconds, and we added 50ms to simulate additional system costs such as network delay. For instance, if t equals 50ms, the

total waiting time would be 100ms, resulting in $S = 1$, which indicates high user satisfaction. Conversely, if t is greater than 150ms, the total waiting time would be over 200ms, and S would be less than 0.5, indicating low user satisfaction.

From the result shown in Figure 5. we can observe that: (1) In general, the MRR of RR/R2PS increases with the larger number of retrieved codes k, but the rate of increase slows down with higher k values. This suggests that retrieving more codes can enhance performance, but the improvement becomes less significant as more codes are retrieved. (2) The performance of R2PS is better than R2 in most cases with different k and different datasets, demonstrating the effectiveness of our proposed probabilistic hard negative sampling strategy for code search. This strategy helps in selecting better codes for training the model, resulting in improved performance. (3) User satisfaction decreases as k increases due to the longer forward propagation time required by the cross-encoder. To achieve higher user satisfaction, it is recommended to use a smaller value of k. (4) The performance of R2PS declines slightly in the CSN-python, CSN-Java, and AdvTest datasets with large k values. With fewer retrieved codes, the candidate codes for the cross-encoder are more difficult, whereas with more retrieved codes, the candidate codes are easier. While hard negative sampling in R2PS enhances the model's ability to handle hard negative samples, its ability to distinguish easier codes may decrease. This explains the decrease in R2PS performance with large k values. (5) Setting k less than 10 causes a significant decline in performance, while k greater than 20 causes a substantial decrease in user satisfaction. Therefore, we recommend setting k between 10 and 20 in our RR/P2PS method.

E. Rank Distribution Analysis (RQ4)

We conduct experiments to show whether our RR/R2PS patch can help the PLMs to rank the relevant codes to higher position. Specifically, we figure out the distribution of ranks of relevant codes in the test set, as shown in Figure 6. We can observe: Compared to UniXcoder, our RR/R2PS patch can rank a larger number of relevant codes to top-1 position, and R2PS patch performs better than RR from this point. Larger number of relevant codes in the top-1 position means user can find useful information within top-1 result in the list. Thus, our RR/R2PS can help the PLMs become the more effective code search tool. (2) We find that the rank distributions of the two CSN datasets (Python and Java) are similar, while are different from StaQC and AdvTest. The steady improvements of our RR/R2PS patch on these different datasets further validate the universality of our proposed method.

V. CONCLUSION

In this paper, our aim is to enhance the performance of PLMs for the code search task during the finetune stage. To achieve this, we introduce several novel approaches. Firstly, we propose a cross-encoder for code search, which enhances the model's capabilities compared to the typical dual-encoder. Next, we propose a retriever and ranker framework for code search that balances both effectiveness and efficiency. Finally, we propose a probabilistic negative sampling method to further improve the retriever and ranker framework's effectiveness. We conducted thorough experiments and found that our R2PS system significantly improves performance while incurring an acceptable extra computing cost.

In light of our findings, we propose two potential future directions for research. Firstly, while the probabilistic sampling (PS) method we used for negative sampling proved to be effective in our experiments, it was relatively simple as it is only based the relevance scores of the dual-encoder. We suggest exploring more sophisticated and reasonable sampling methods to further enhance the model's performance. Secondly, while our paper focused solely on the code search task for code PLMs, these models can also be used for other downstream tasks such as code generation and bug fixing. We believe that exploring finetune methods for code PLMs in other downstream tasks would be a meaningful area of research.

REFERENCES

[1] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 933–944.

[2] J. Cambronero, H. Li, S. Kim, K. Sen, and S. Chandra, "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*, M. Dumas, D. Pfahl, S. Apel, and A. Russo, Eds. ACM, 2019, pp. 964–974.

[3] X. Ling, L. Wu, S. Wang, G. Pan, T. Ma, F. Xu, A. X. Liu, C. Wu, and S. Ji, "Deep graph matching and searching for semantic code retrieval," *ACM Trans. Knowl. Discov. Data*, vol. 15, no. 5, pp. 88:1–88:21, 2021.

[4] T. Diamantopoulos, G. Karagiannopoulos, and A. L. Symeonidis, "Code-catch: 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*, W. F. Tichy and L. L. Minku, Eds. ACM, 2018, pp. 21–27.

[5] S. Luan, D. Yang, C. Barnaby, K. Sen, and S. Chandra, "Aroma: code recommendation via structural code search," *Proc. ACM Program. Lang.*, vol. 3, no. OOPSLA, pp. 152:1–152:28, 2019.

[6] B. Sisman and A. C. Kak, "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*, T. Zimmermann, M. D. Penta, and S. Kim, Eds. IEEE Computer Society, 2013, pp. 309–318.

[7] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, M. Tufano, S. K. Deng, C. B. Clement, D. Drain, N. Sundaresan, J. Yin, D. Jiang, and M. Zhou, "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, 2021.

[8] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "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*, ser. Findings of ACL, T. Cohn, Y. He, and Y. Liu, Eds., vol. EMNLP 2020. Association for Computational Linguistics, 2020, pp. 1536–1547.

[9] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "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*, S. Muresan, P. Nakov, and A. Villavicencio, Eds. Association for Computational Linguistics, 2022, pp. 7212–7225.
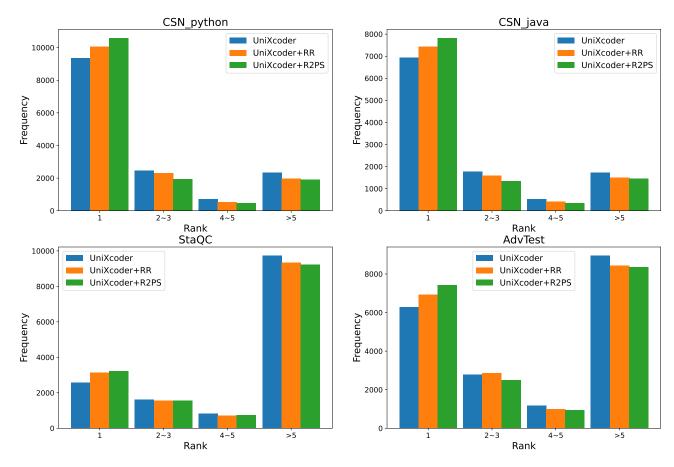
Fig. 6. The distribution of the ranks of relevant codes in the test set.

[10] J. Zhang, S. Panthaplackel, P. Nie, J. J. Li, and M. Gligoric, "Coditt5: Pretraining for source code and natural language editing," in *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*. ACM, 2022, pp. 22:1–22:12. [Online]. Available: https://doi.org/10.1145/3551349.3556955

[11] C. Niu, C. Li, V. Ng, J. Ge, L. Huang, and B. Luo, "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*. ACM, 2022, pp. 1–13.

[12] J. Guo, Y. Fan, Q. Ai, and W. B. Croft, "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*, S. Mukhopadhyay, C. Zhai, E. Bertino, F. Crestani, J. Mostafa, J. Tang, L. Si, X. Zhou, Y. Chang, Y. Li, and P. Sondhi, Eds. ACM, 2016, pp. 55–64.

[13] C. Xiong, Z. Dai, J. Callan, Z. Liu, and R. Power, "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*, N. Kando, T. Sakai, H. Joho, H. Li, A. P. de Vries, and R. W. White, Eds. ACM, 2017, pp. 55–64.

[14] Y. Luan, J. Eisenstein, K. Toutanova, and M. Collins, "Sparse, dense, and attentional representations for text retrieval," *Trans. Assoc. Comput. Linguistics*, vol. 9, pp. 329–345, 2021.

[15] S. Sachdev, H. Li, S. Luan, S. Kim, K. Sen, and S. Chandra, "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*, J. Gottschlich and A. Cheung, Eds. ACM, 2018, pp. 31–41.

[16] N. D. Q. Bui, Y. Yu, and L. Jiang, "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*, F. Diaz, C. Shah, T. Suel, P. Castells, R. Jones, and T. Sakai, Eds. ACM, 2021, pp. 511–521.

[17] Y. Wan, J. Shu, Y. Sui, G. Xu, Z. Zhao, J. Wu, and P. S. Yu, "Multimodal 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*. IEEE, 2019, pp. 13–25.

[18] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "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)*, J. Burstein, C. Doran, and T. Solorio, Eds. Association for Computational Linguistics, 2019, pp. 4171–4186.

[19] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.

[20] X. Li, Y. Gong, Y. Shen, X. Qiu, H. Zhang, B. Yao, W. Qi, D. Jiang, W. Chen, and N. Duan, "Coderetriever: Large-scale contrastive pretraining for code search," in *the Association for Computational Linguistics: EMNLP 2022*, 2022.

[21] X. Wang, Y. Wang, Y. Wan, J. Wang, P. Zhou, L. Li, H. Wu, and

J. Liu, "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*, M. Carpuat, M. de Marneffe, and I. V. M. Ruíz, Eds. Association for Computational Linguistics, 2022, pp. 1066–1077.

[22] M. Lachaux, B. Rozière, M. Szafraniec, and G. Lample, "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*, M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021, pp. 14 967–14 979.

[23] W. U. Ahmad, S. Chakraborty, B. Ray, and K. Chang, "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*, K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tür, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, and Y. Zhou, Eds. Association for Computational Linguistics, 2021, pp. 2655–2668.

[24] J. Huang, D. Tang, L. Shou, M. Gong, K. Xu, D. Jiang, M. Zhou, and N. Duan, "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*, C. Zong, F. Xia, W. Li, and R. Navigli, Eds. Association for Computational Linguistics, 2021, pp. 5690–5700.

[25] C. Zeng, Y. Yu, S. Li, X. Xia, Z. Wang, M. Geng, L. Bai, W. Dong, and X. Liao, "degraphcs: Embedding variable-based flow graph for neural code search," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 2, pp. 34:1–34:27, 2023.

[26] H. Li, C. Miao, C. Leung, Y. Huang, Y. Huang, H. Zhang, and Y. Wang, "Exploring representation-level augmentation for code search," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, Y. Goldberg, Z. Kozareva, and Y. Zhang, Eds. Association for Computational Linguistics, 2022, pp. 4924–4936.

[27] Y. Hao, L. Dong, F. Wei, and K. Xu, "Self-attention attribution: Interpreting information interactions inside transformer," in *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 2021, pp. 12 963–12 971.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "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*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 5998–6008.

[29] A. van den Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *CoRR*, vol. abs/1807.03748, 2018.

[30] K. He, H. Fan, Y. Wu, S. Xie, and R. B. Girshick, "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*. Computer Vision Foundation / IEEE, 2020, pp. 9726–9735.

[31] H. Husain, H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet challenge: Evaluating the state of semantic code search," *CoRR*, vol. abs/1909.09436, 2019.

[32] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. B. Clement, D. Drain, D. Jiang, D. Tang, G. Li, L. Zhou, L. Shou, L. Zhou, M. Tufano, M. Gong, M. Zhou, N. Duan, N. Sundaresan, S. K. Deng, S. Fu, and S. Liu, "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*, J. Vanschoren and S. Yeung, Eds., 2021.

[33] Z. Yao, D. S. Weld, W. Chen, and H. Sun, "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*, P. Champin, F. Gandon, M. Lalmas, and P. G. Ipeirotis, Eds. ACM, 2018, pp. 1693–1703.

[34] X. Wang, Y. Wang, P. Zhou, F. Mi, M. Xiao, Y. Wang, L. Li, X. Liu, H. Wu, J. Liu, and X. Jiang, "SyncoBERT: contrastive learning for syntax enhanced code pre-trained model," *CoRR*, vol. abs/2108.04556, 2021.