SPENCER: Self-Adaptive Model Distillation for Efficient Code Retrieval

Wenchao Gu, Zongyi Lyu, Yanlin Wang, Hongyu Zhang, Cuiyun Gao, Michael R. Lyu

Abstract—Code retrieval aims to provide users with desired code snippets based on users' natural language queries. With the development of deep learning technologies, adopting pre-trained models for this task has become mainstream. Considering the retrieval efficiency, most of the previous approaches adopt a dual-encoder for this task, which encodes the description and code snippet into representation vectors, respectively. However, the model structure of the dual-encoder tends to limit the model's performance, since it lacks the interaction between the code snippet and description at the bottom layer of the model during training. To improve the model's effectiveness while preserving its efficiency, we propose a framework, which adopts **S**elf-Ada**P**tive Model Distillation for **E**fficient **C**od**E** **R**etrieval, named SPENCER. SPENCER first adopts the dual-encoder to narrow the search space and then adopts the cross-encoder to improve accuracy. To improve the efficiency of SPENCER, we propose a novel model distillation technique, which can greatly reduce the inference time of the dual-encoder while maintaining the overall performance. We also propose a teaching assistant selection strategy for our model distillation, which can adaptively select the suitable teaching assistant models for different pre-trained models during the model distillation to ensure the model performance. Extensive experiments demonstrate that the combination of dual-encoder and cross-encoder improves overall performance compared to solely dual-encoder-based models for code retrieval. Besides, our model distillation technique retains over 98% of the overall performance while reducing the inference time of the dual-encoder by 70%.

Index Terms—Code retrieval, Deep learning, Model distillation

1 INTRODUCTION

With the advancement of Internet technology and the rise of open-source communities, utilizing the web to search for necessary code has become a prevailing trend among developers [1], [2]. A significant challenge for the effective search lies in the semantic gap between human natural language and programming languages. To mitigate the gap, extensive efforts [3], [4], [5] have been devoted to accurately retrieve the required code through natural language.

With the rapid development of neural network technology and pre-training methods, fine-tuning pre-trained code-based models has become the prevailing approach in the code retrieval task. Most of pre-trained model based approaches adopt the dual-encoder [6], [7]. As shown in Fig. 1, code snippets and natural language-based descriptions are encoded separately by two independent encoders in the dual-encoder based approaches [6], [7]. The similarity between the encoded representation vectors of the code and description is then computed using cosine similarity.

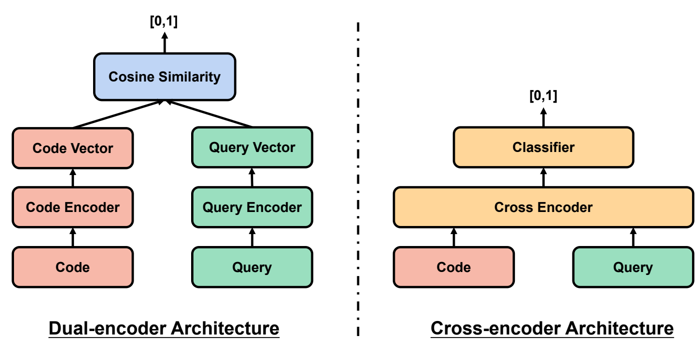


Fig. 1. Illustration of the code retrieval approach with dual-encoder architecture and cross-encoder architecture.

However, the lack of interaction between the code snippet and description at the bottom layer of the model during the training limits the model performance [8].

To involve the interaction between the code snippet and description, one commonly used solution is the cross-encoder [9]. As shown in Fig. 1, the architecture incorporates both code snippets and natural language descriptions as a single model input. This unified approach generates a normalized score that evaluates the alignment between the provided code and its corresponding description, effectively quantifying their similarity degree. Nevertheless, the cross-encoder does face efficiency challenges. In contrast to the dual-encoder, which can compute and store the code's representation vector in advance within a database, the cross-encoder lacks this precomputation capability. This stems from the cross-encoder's reliance on input from both the query and the code, leading to the recalculation of matching

- *Wenchao Gu and Michael R. Lyu are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China.*
E-mail: wcgu@cse.cuhk.edu.hk, lyu@cse.cuhk.edu.hk
- *Zongyi Lyu and Cuiyun Gao are with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China.*
E-mail: 200110118@stu.hit.edu.cn, gaocuiyun@hit.edu.cn
- *Yanlin Wang is with the Software Engineering School, Sun Yat-sen University, Zhuhai, China.*
E-mail: yanlin-wang@outlook.com
- *Hongyu Zhang is with the School of Big Data and Software Engineering, Chongqing University, Chongqing, China.*
E-mail: hongyujohn@gmail.com

Manuscript received September 2023

scores between the query and every code entry in the database. Therefore, the inference cost associated with the cross-encoder becomes impractical when dealing with large code databases.

To obtain the high performance of the cross-encoder as much as possible while considering the efficiency problem, we propose a novel framework named SPENCER for the task of code retrieval. This framework adopts the dual-encoder initially to select several candidates from the entire database based on the given query. Once the candidates are retrieved, the query is sequentially combined with each candidate and passed into the cross-encoder. This process calculates the matching score for each selected pair and re-ranks the order of candidates. The adoption of this approach can greatly reduce the inference cost of the cross-encoder from the entire database to a small fixed number.

Since the primary purpose of the dual-encoder within our framework is to select a fixed number of candidates, it is sufficient to ensure that the correct answer is among the returned candidates. Nevertheless, employing a pre-trained model as the dual-encoder comes with a high computation cost due to its large size. We argue that such a large model might not be necessary for the dual-encoder's function and the approaches for reducing the dual-encoder's model size while simultaneously upholding its performance within this framework still remain unexplored. To address this problem, we propose a novel model distillation approach for the dual-encoder on the query side. The proposed model distillation approach makes our distilled query encoder learn the similarity in both single modality and dual modality from the teacher dual-encoder without relying on ground-truth information. Such a model distillation approach enables us to achieve an efficient and accurate dual-encoder without sacrificing too much performance.

To further improve the performance of distilled query encoder, we propose a self-adaptive teaching assistant selection approach for our model distillation. This approach can dynamically select suitable teaching assistants for the distilled query encoder during the model distillation process.

We conducted comprehensive experiments to validate the effectiveness of our proposed framework, incorporating the proposed model distillation approach. The results of the experiments demonstrate the efficiency of the framework in significantly improving performance with considerable computation costs. Our model distillation approach is highly effective, reducing the inference time by around 70% of the dual encoder model while preserving more than 98% of overall performance.

We summarize the main contributions of this paper as follows:

- In this study, we present a framework that leverages both the dual-encoder and cross-encoder components of pre-trained models for the code retrieval task. The experimental results demonstrate that this integrated approach can achieve the high accuracy associated with the cross-encoder.
- We present a novel approach to model distillation for the dual-encoder within a unified framework. Our method greatly reduces the parameters of the dual-encoder while preserving the most performance of the unified framework.
- We present a novel approach to select the assist model during the process of model distillation, which can adaptively select the suitable assistant model for different pre-trained models during training. This adaptive selection process aims to enhance performance and achieve better overall results.

The remainder of this paper is structured as follows: Section 2 provides an overview of the architecture of our proposed SPENCER, including the design of the unified framework, and the design of the model distillation approach with the teaching assistant. Section 3 describes our experimental setup, including the datasets used, evaluation metrics, and implementation specifics. In Section 4, we present the experimental results and provide our analysis. In Section 5, we discuss the threats to the validity of our experiments. Section 6 discusses the related work on code retrieval and knowledge distillation, while Section 7 concludes the paper.

2 METHODOLOGY

In this section, we first present the principles of our proposed framework. Then we explain the training strategy for both the dual-encoder and cross-encoder. Next, we will introduce the model distillation design for the dual-encoder. Finally, we will introduce our self-adaptive approach for identifying a suitable teaching assistant during the model distillation process.

2.1 Overview

Fig. 2 illustrates the overall framework of our approach. In this framework, both the dual-encoder and cross-encoder are trained in advance. After training, code snippets inside the code database are encoded into code representation vectors using the code-encoder, which is a dual-encoder. These code vectors are then pre-stored in the code database. When a user query is received, the query encoder, which is another kind of dual-encoder, processes the query and generates a query representation vector. To find the most relevant code candidates, the cosine similarity between the query vector and each code vector in the database is calculated and sorted in descending order. The top K code candidates with the highest cosine similarity would be retrieved. Each of these candidates is then combined with the original query input, resulting in a new concatenated input. These new inputs are then fed into the cross-encoder, and the code candidates are re-ranked based on a descending sort of matching scores obtained from the cross-encoder. Finally, the top K re-ranked code list is concatenated with the remaining part of the code list from the dual-encoder. This combined list is considered the final code list and returned to the user.

2.2 Dual-Encoder Training

Under our proposed framework, the functionality of the dual-encoder is to select the top K code candidates that are most likely to contain the correct answer. There are dual-encoders in our framework: the query encoder and the

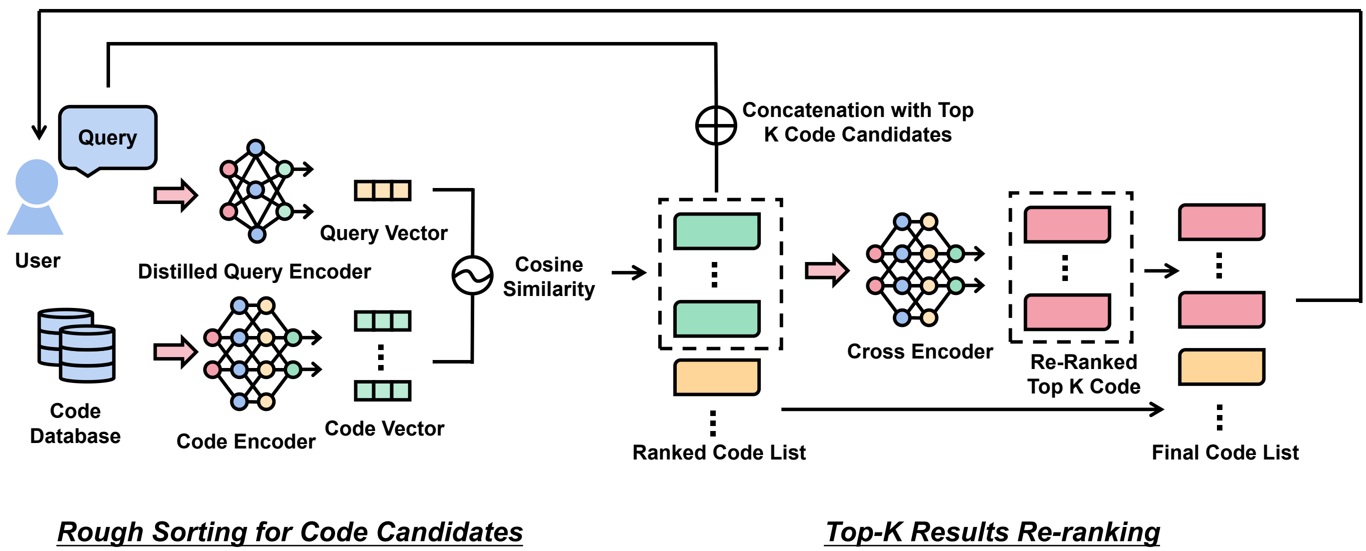


Fig. 2. The overall framework of SPENCER. Code retrieval under this framework can be split into two steps: rough sorting for code candidates and top-K results re-ranking. **Rough sorting for code candidates:** the code snippets in the code database and the given query are embedded into vectors via the dual-encoder, respectively. Then the cosine similarity between the query vector and code vector will be calculated and the code candidates will be sorted in a descending order according to this similarity. **Top-K results re-ranking:** The top-K code snippets in the previous code candidates list are concatenated with the given query and the new input will be fed into cross-encoder. The top-K results in the code candidates list will be re-ranked according to the match score from the cross-encoder.

code encoder. Both of these encoders utilize Transformer-based pre-trained models. To prepare the input data for the encoders, the queries and code snippets are tokenized into sequences of tokens. For each sequence, a special token, denoted as $[CLS]$, is added at the beginning, resulting in a token sequence with the form $[CLS], [Tok1], [Tok2], ...$ During the training process of the dual-encoders, the token sequence of the code and the query are fed into the code encoder and the query encoder, respectively. We extract the hidden vectors of the first token from the last layer in the query encoder and the code encoder, which serves as the code representation vector and the query representation vector, respectively. To align the code vectors and query vectors effectively, we employ contrastive learning during the training phase. The loss for the dual-encoder training consists of three components: the contrastive loss for the code modality, the contrastive loss for the query modality, and the contrastive loss for the cross-modality. The contrastive loss for the code modality is formed as follows:

$$\mathcal{L}_C = -\sum_{i=1}^{n} log \frac{exp(c_i \cdot c_i^+/\tau)}{\sum_{j=1, i\neq j}^{n} exp(c_i \cdot c_j^-/\tau)} \tag{1}$$

where c_i^+ is the positive sample of the i-th code snippet, c_j^- is the negative sample of the j-th code snippet, n is the size of training batch and τ is the temperature parameter. We adopt different mask to encoder with the same input to generate the positive samples, which is followed by SimCSE [10].

Similarly, the contrastive loss for the query modality is:

$$\mathcal{L}_Q = -\sum_{i=1}^{n} log \frac{exp(q_i \cdot q_i^+/\tau)}{\sum_{j=1, i\neq j}^{n} exp(q_i \cdot q_j^-/\tau)} \tag{2}$$

where q_i^+ is the positive sample of the i-th description, q_j^- is the negative sample of the j-th description, n is the size of training batch and τ is the temperature parameter. The method of positive sample generation is the same as the previous one.

The contrastive loss for the cross modality is

$$\mathcal{L}_D = -\sum_{i=1}^{n} log \frac{exp(c_i \cdot q_i/\tau)}{\sum_{j=1, i\neq j}^{n} exp(c_i \cdot q_j/\tau)} \tag{3}$$

where c_i is the i-th code snippet, q_i is the i-th description, n is the size of training batch and τ is the temperature parameter. In the contrastive loss for the cross modality, the corresponding description is adopted as the positive sample for the given code and the unmatched description is adopted as the negative sample for the given code.

The total loss for the dual-encoder training is shown as:

$$\mathcal{L} = \mathcal{L}_C + \mathcal{L}_Q + \mathcal{L}_D \tag{4}$$

2.3 Cross-Encoder Training

In this framework, the role of the cross-encoder is to re-rank the selected code candidates from the dual-encoder, aiming for accuracy improvement. For the cross-encoder, the code and query will be firstly tokenized into a token sequence, respectively. Then these two token sequences will be combined into a single token sequence. $[CLS]$ will be added at the beginning of the toke sequence and $[SEP]$ will be added between the code token sequence and query token sequence. The token sequence will be $[CLS], [Code\_tok1], ..., [SEP], [Query\_tok1], ...$ The cross-encoder is trained using the cross-entropy loss, which is defined as follows:

$$\mathcal{L} = -\sum_{i=1}^{n} (ylog\hat{y} + (1-y)log(1-\hat{y})) \tag{5}$$

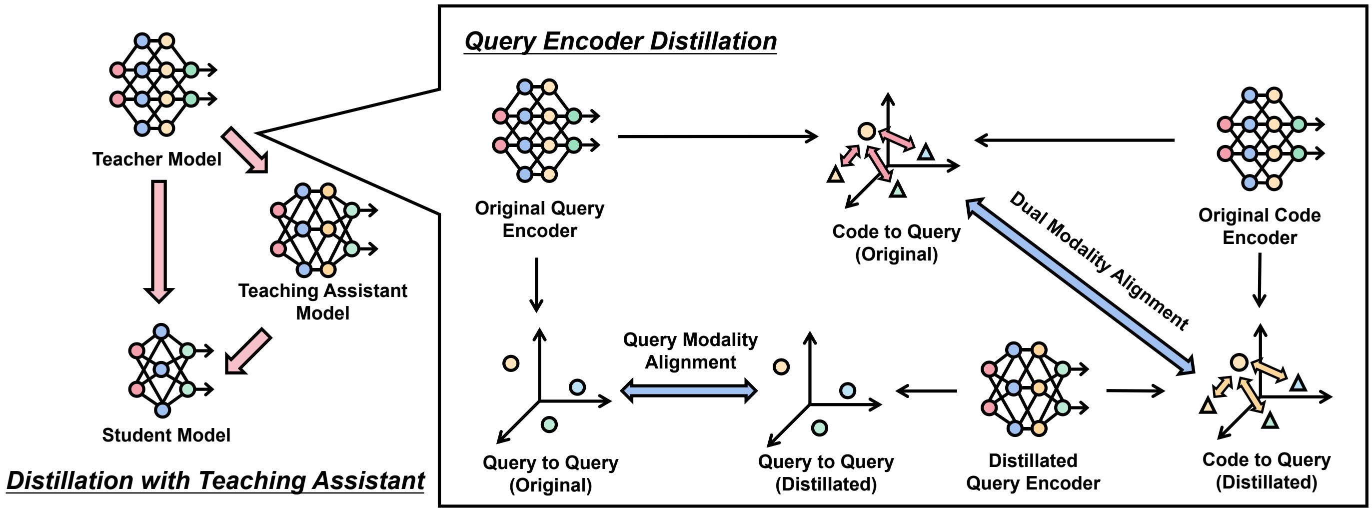


Fig. 3. The process of model distillation within our framework. There are two main components in our model distillation, which are Distillation with Teaching Assistant and Query Encoder Distillation. **Distillation with Teaching Assistant:** To alleviate the learning problem brought by the large size gap between the large teacher model and the small student model, a middle teaching assistant model will be trained at first. Then both the teacher model and teaching assistant model will be utilized for the student model training. The details of the selection strategy for the teaching assistant model will be introduced in the following section. **Query Encoder Distillation:** A small query encoder will be distilled from the original query encoder with the training loss of both single modality and dual modality. The single modality loss aims to align the output of the distillated query encoder to the output of the original query encoder. The purpose of the dual-modality loss is to provide the relative positional relationship between the output from both the original query encoder and code encoder for the distillated query encoder learning.

where y is the ground-truth label which indicates whether the given pair of query and code is matched and $\hat{y}$ is the normalized prediction score from the cross-encoder.

2.4 Query Encoder Distillation

In the dual-encoder, the code encoder's primary role is to encode code snippets into code representation vectors during the construction of the code database. Once this encoding process is complete, the code encoder remains inactive until new code snippets are added to the database. Unlike the code encoder, the query encoder will be invoked when the system receives the query from the user. Since the model distillation will unavoidably lead to performance degradation, it is better to keep the model unchanged if the model will not affect the efficiency of the entire system. Therefore, our focus shifts to the model distillation for the query encoder. Unlike previous approaches that attempted to distill the distribution of logits output from the model in the code area, our objective is to distill the high-dimensional spatial projection capabilities of pre-trained models into smaller models. Our goal is to preserve the performance of the pre-trained model as much as possible. To achieve this, we propose a novel distillation loss for our model distillation. Fig. 3 illustrates the process of query encoder distillation. The distillation loss comprises two components: one for the query modality and another for the dual modality. The distillation loss for the query modality is outlined below:

$$\mathcal{L}_{\mathcal{Q}} = \sum_{i=1}^{n}(1 - \frac{\hat{q}_i \cdot q_i}{||\hat{q}_i|| \cdot ||q_i||}) \tag{6}$$

where q_i is the i-th code representation vector from the target model, which is the model needs to be distilled, and $\hat{q}_i$ is the i-th code representation vector from our distilled model. Since we hope that the representation vector from the distilled model can be identical to the representation vector from the target model, the cosine similarity between these two representation vectors and align the similarity with 1.

However, the performance of the distilled model will drop significantly if the distilled model capacity has a large gap with the target model capacity. To tackle this issue, the distillation loss of dual-modality is introduced and it is shown below:

$$\mathcal{L}_{\mathcal{D}} = \sum_{i=1}^{n}|\frac{c_i \cdot q_i}{||c_i|| \cdot ||q_i||} - \frac{\hat{q}_i \cdot c_i}{||\hat{q}_i|| \cdot ||c_i||}| \tag{7}$$

where c_i is the i-th code representation vector from the original code encoder, q_i is the i-th query representation vector from the target query encoder, and $\hat{q}_i$ is the i-th query representation vector from the distillated query encoder. The cosine similarity serves as the primary metric in code retrieval. By considering the similarity of the dual modality as part of the training target, we preserve the relative positional relationship between queries and codes, leading to enhanced performance of the distilled model.

In contrast to the conventional distillation approach used in classification tasks, where the ground-truth labels serve as the training target, our findings indicate that incorporating ground-truth labels during the query encoder distillation does not contribute to the enhancement of model performance; on the contrary, it negatively impacts the performance. The specifics will be discussed in the upcoming section.

Algorithm 1 Algorithm for Self-Adaptive Teaching Assistant Selection

Input: CM_T: Original teacher model for the code encoding, QM: Original model for the query encoding, P: The Reduced parameters for every step, T: Threshold for the performance drop
Output: $CM_{Student}$: Student model for the code encoding
$CM_{A1} \leftarrow CM_T$
$CM_{A2} \leftarrow \text{ModelCompression}(CM_T, P)$
$CM_{A2} \leftarrow \text{ModelDistillation}(CM_{A1}, QM)$
$CM_S \leftarrow CM_{A2}$
$Score_T = \text{Validation}(CM_T, QM)$
$Score_S = \text{Validation}(CM_S, QM)$
while $Score_T - Score_S < T$ **do**
  $CM_{Temp1} \leftarrow \text{ModelCompression}(CM_{A2}, P)$
  $CM_{Temp2} \leftarrow \text{ModelCompression}(CM_{A2}, P)$
  $CM_{Temp1} \leftarrow \text{ModelDistillation}(CM_{A1}, QM)$
  $CM_{Temp2} \leftarrow \text{ModelDistillation}(CM_{A2}, QM)$
  $Score_{A1} = \text{Validation}(CM_{Temp1}, QM)$
  $Score_{A2} = \text{Validation}(CM_{Temp2}, QM)$
  if $Score_{A1} > Score_{A2}$ **then**
    $CM_{A2} \leftarrow CM_{Temp1}$
    $Score_S = Score_{A1}$
  else
    $CM_{A1} \leftarrow CM_{A2}$
    $CM_{A2} \leftarrow CM_{Temp1}$
    $Score_S = Score_{A2}$
  if $Score_T - Score_S < T$ **then**
    $CM_S \leftarrow CM_{A2}$
return CM_S

2.5 Self-Adaptive Teaching Assistant Selection

Addressing the challenge of a large capability gap between teacher and student models, a popular approach involves introducing a teaching assistant model. This intermediary model, with a size between that of the teacher and student models, helps bridge the knowledge gap. Initially, the teaching assistant model learns from the teacher model, converting complex knowledge into a more accessible form. Subsequently, the student model learns from the teaching assistant model, enhancing its grasp of the teacher's knowledge. However, determining the optimal size for the teaching assistant model poses difficulties. The vast search space and associated computational costs make exhaustive exploration impractical. Additionally, even models with the same architecture may require different-sized teaching assistant models due to varying pre-trained models. Therefore, selecting the appropriate teaching assistant model for different pre-trained models becomes a crucial challenge.

To address this problem, we propose a novel approach for selecting the teaching assistant model during the model distillation process. Our method dynamically adjusts the teaching assistant model to find the most suitable one. The detailed steps of our proposed approach can be found in Algorithm 1. In Algorithm 1, $\text{ModelCompression}(\text{M}, \text{P})$ indicates the model compression operation for the model M with reduced parameter P. $\text{ModelDistillation}(\text{M}_1, \text{M}_2)$ indicates the model distillation operation from the M_1 with the reference of M_2. $\text{Validation}(\text{M}_1, \text{M}_2)$ indicates the performance validation from the model M_1 and M_2. For the initialization, we will set the amount of model parameters to be reduced in each distillation step and train a teaching assistant model with reduced parameters at first. Subsequently, we consider the original teacher model and the newly distilled teaching assistant as two target models for the distillation. The distillation process yields two student models from these targets, which are then compared in terms of their performance. The student model with superior performance replaces the teacher model that teaches the less competent student. This replacement is necessary because the teacher model exhibiting poor teaching ability is identified through this comparison. The two new target models (the better student model and the original teacher model) are then used for distilling a new student model. This loop of distillation continues until the model reaches its minimum size, or the performance difference between the student model and the original model exceeds a preset threshold value. The final student model, resulting from this iterative process, will be preserved and employed as the code encoder in our proposed framework.

By following this approach, we can select the suitable teaching assistant model for different pre-trained models to further improve the performance of the distilled model.

TABLE 1
Dataset statistics.

Dataset	Training	Validation	Test
Python	412,178	23,107	22,176
Java	454,451	15,328	26,909

3 Experimental Settings

3.1 Datasets

The dataset we utilized to evaluate the proposed framework is initially from CodeBERT. CodeBERT selects the descriptions and code snippets from CodeSearchNet. It makes matched code snippets and descriptions as positive pairs and makes the unmatched code snippets and descriptions as negative pairs. We directly utilize the dataset from the CodeBERT to train the cross-encoder. As for the training of dual-encoder in our proposed framework, we reorganized the dataset from CodeBERT. The reason why we reorganized the dataset is the difference in training mechanism between the dual-encoder and cross-encoder. Cross-encoder treats the code retrieval task as the classification task and the inputs are the pairs of code snippets and queries. On the contrary, the dual-encoder treats the code retrieval task as the data projection task which projects the code snippets and queries into the same high dimensional space. Similar code snippets and queries should be close to each other and dissimilar code snippets and queries should be far from each other. Due to the difference in mechanisms, the dual-encoder only accepts the data from code modality or query modality. To address this different data format requirement, we only keep the positive pairs of code snippets and queries and remove all the negative pairs from the dataset. The reason we remove the negative pairs is that the training of dual-encoder needs the label for the alignment of code snippets and queries but negative pairs cannot provide such labels. Table 1 shows the statistics of the datasets.

3.2 Baselines

We select three state-of-the-art Transformer-based pre-trained models in the code area to validate the effectiveness of our proposed framework, which are shown below:

- **CodeBERT** is a pre-trained model based on a Transformer with 12 layers. It combines code snippets and descriptions, converts them as token sequences, and utilizes them as the input of the model.
- **GraphCodeBERT** is another pre-trained Transformer based model. Unlike CodeBERT which only utilizes the token sequence as the input, GraphCodeBERT also considers the data flow of code snippets and utilizes it as the additional input.
- **CodeT5** is a pre-trained Transformer-based model with both an encoder and a decoder. CodeT5 is pre-trained with three identifier-aware pre-training tasks, which lead to the ability to recover masked identifiers in the code.

3.3 Metrics

$R@k$ (recall at k) and MRR (mean reciprocal rank) are utilized as the evaluation metrics to evaluate the performance of our proposed framework. $R@k$ is the metric to evaluate whether the model can return the correct answer within top K candidates. It is widely used to evaluate the performance of the code retrieval models in previous research [11], [12], [13], [14]. The definition of $R@k$ is shown below:

$$R@k = \frac{1}{|Q|}\sum_{q=1}^{Q}\delta(FRank_q \leq k), \tag{8}$$

where Q denotes the query set and $FRank_q$ is the rank of the correct answer for query q. $\delta(Frank_q \leq k)$ returns 1 if the correct result is within the top k returning results, otherwise it returns 0. The higher $R@k$ is, the better performance the model has.

MRR is a popular metric used in recommendation systems and it is also widely used to evaluate the performance in the task of code retrieval [7], [9]:

$$MRR = \frac{1}{|Q|}\sum_{q=1}^{Q}\frac{1}{FRank_q} \tag{9}$$

Similar to $R@k$, a higher MRR indicates better performance.

3.4 Implementation Details

In our experiment, we fine-tuned the dual-encoder and cross-encoder sourced from pre-trained models available in the public repository. These original encoders are based on the Transformer model, consisting of 12 layers and 12 heads. The dimension of the output vectors from the dual-encoder is 768. CodeBERT and GraphCodeBERT are encode-only models and CodeT5 is an encoder-decoder model. To ensure equitable comparison of experimental results, we omitted the decoder of CodeT5 and exclusively employed its encoder. This decision was made because our task focuses solely on encoding code and queries into representation vectors. In addition, we set the maximum input length for our models as 512 and the dropout ratio as 0.2. For model optimization, we maintained a consistent learning rate of 1e-5 across all models. The optimization process employed the AdamW algorithm [15]. We trained our models on a server with Tesla A100 and we trained both dual-encoders and cross-encoders for 8 epochs. The training batch size we used is 16. An early stopping strategy is adopted to avoid over-fitting for all models. The training batch size for the model training is 16.

In each step of the distillation process, three layers were eliminated. The hyperparameter T in our teaching assistant selection algorithm was set to 0.01.

In our experiment, we partitioned the test dataset into distinct search pools whose size is 1,000 for evaluation purposes. The models were evaluated in each pool and the average results from all the pools are reported in our paper.

4 Evaluation

4.1 RQ1: The effectiveness of our proposed framework

Table 2 illustrates the experiment results of the overall performance comparison of different encoders with different pre-trained models. $\texttt{Model}_{\texttt{Dual}}$ represents the approach which only adopts the dual-encoder for code retrieval. $\texttt{Model}_{\texttt{SPENCERnoDistill}}$ indicates the code retrieval approach which adopts our proposed framework but the model distillation part is removed. $\texttt{Model}_{\texttt{SPENCER}}$ represents the code retrieval approach that adopts the complete version of our proposed framework SPENCER.

From the experiment results in Table 2, we can find that the performance improvement of $\texttt{Model}_{\texttt{SPENCERnoDistill}}$ will be affected by the selection of the pre-trained model. Specifically, the performance improvement with the pre-trained model named CodeT5 is around 200% as the improvement with CodeBERT or GraphCodeBERT on the metric of R@1 on both datasets. Besides, compared to $\texttt{Model}_{\texttt{SPENCERnoDistill}}$, we can find that $\texttt{Model}_{\texttt{SPENCER}}$ can preserve most of the performance. Specifically, $\texttt{Model}_{\texttt{SPENCER}}$ can preserve more than 98% performance of $\texttt{Model}_{\texttt{SPENCERnoDistill}}$ on all the metrics with both datasets. Here we need to pay attention that the performance of $\texttt{Model}_{\texttt{SPENCER}}$ on the metric of R@5 is even worse than the performance of $\texttt{Model}_{\texttt{Dual}}$. The reason is that the code candidates are recalled by the dual-encoder in our proposed framework. Since we set the recall number as 5 in our experiments, the performance on the metric R@5 is directly determined by the dual-encoder and the cross-encoder does not make any contribution to the performance improvement on this metric. The distillation of the model will lead to the loss of the performance so that the overall performance of $\texttt{Model}_{\texttt{SPENCER}}$ dropped on the metric R@5.

Table 3 showcases the inference time costs associated with the query encoder within our proposed framework. Distilled refers to the inference time cost of the distilled query encoder, while Original refers to the inference time cost of the original query encoder. The experimental results reveal that our distillation approach greatly diminishes the inference time of the query encoder within our framework by approximately 70%. These results demonstrate the effectiveness of our proposed distillation methods in enhancing the model efficiency.

TABLE 2
Results of overall performance comparison with different pre-trained models. The percentage of performance improvement is calculated based on the performance of the dual encoder. $\texttt{Model}_{\texttt{Dual}}$ indicates the approach which only adopts the dual-encoder. $\texttt{Model}_{\texttt{SPENCERnoDistill}}$ indicates the approach which adopts SPENCER but the model distillation part is removed. $\texttt{Model}_{\texttt{SPENCER}}$ indicates the approach which adopts the complete version of SPENCER.

Model	Python				Java			
	R@1	R@3	R@5	MRR	R@1	R@3	R@5	MRR
$\texttt{CodeBERT}_{\texttt{Dual}}$	0.652	0.839	0.888	0.757	0.533	0.704	0.754	0.633
$\texttt{CodeBERT}_{\texttt{SPENCERnoDistill}}$	0.714 (↑9.5%)	0.865 (↑3.1%)	0.888 (0.0%)	0.798 (↑5.4%)	0.575 (↑7.9%)	0.722 (↑2.6%)	0.754 (0.0%)	0.661 (↑4.4%)
$\texttt{CodeBERT}_{\texttt{SPENCER}}$	0.710 (↑8.9%)	0.857 (↑2.1%)	0.879 (↓1.0%)	0.792 (↑4.6%)	0.569 (↑6.8%)	0.711 (↑1.3%)	0.742 (↓1.6%)	0.653 (↑3.2%)
$\texttt{GraphCodeBERT}_{\texttt{Dual}}$	0.669	0.853	0.901	0.771	0.541	0.712	0.760	0.640
$\texttt{GraphCodeBERT}_{\texttt{SPENCERnoDistill}}$	0.727 (↑8.7%)	0.875 (↑2.6%)	0.901 (0.0%)	0.809 (↑4.9%)	0.590 (↑9.1%)	0.750 (↑5.3%)	0.760 (0.0%)	0.671 (↑4.8%)
$\texttt{GraphCodeBERT}_{\texttt{SPENCER}}$	0.721 (↑7.8%)	0.867 (↑1.6%)	0.891 (↓1.1%)	0.802 (↑4.0%)	0.582 (↑7.6%)	0.720 (↑1.1%)	0.749 (↓1.4%)	0.664 (↑3.8%)
$\texttt{CodeT5}_{\texttt{Dual}}$	0.655	0.842	0.892	0.760	0.500	0.681	0.737	0.608
$\texttt{CodeT5}_{\texttt{SPENCERnoDistill}}$	0.757 (↑15.6%)	0.880 (↑4.5%)	0.892 (0.0%)	0.826 (↑8.7%)	0.587 (↑17.4%)	0.718 (↑5.4%)	0.737 (0.0%)	0.664 (↑9.2%)
$\texttt{CodeT5}_{\texttt{SPENCER}}$	0.751 (↑14.7%)	0.870 (↑3.3%)	0.882 (↓1.1%)	0.819 (↑7.8%)	0.579 (↑15.8%)	0.707 (↑3.8%)	0.726 (↓1.5%)	0.656 (↑7.9%)

TABLE 3
Results of inference time cost comparison of the query encoder with different pre-trained models.

Model	Distilled	Original	Ratio
$\texttt{CodeBERT}_{\texttt{Python}}$	15.0s	52.2s	28.7%
$\texttt{CodeBERT}_{\texttt{Java}}$	12.4s	42.2s	29.4%
$\texttt{GraphCodeBERT}_{\texttt{Python}}$	14.2s	51.4s	27.6%
$\texttt{GraphCodeBERT}_{\texttt{Java}}$	12.4s	42.2s	29.4%
$\texttt{CodeT5}_{\texttt{Python}}$	22.0s	69.1s	31.8%
$\texttt{CodeT5}_{\texttt{Java}}$	14.7s	47.8s	30.8%

In summary, SPENCER can effectively improve the code retrieval performance. Besides, our proposed model distillation approach can efficiently reduce the 70% inference time of the query encoder inside our framework while preserving more than 98% overall performance.

4.2 RQ2: The effectiveness of our distillation approach

In this section, we investigate the effectiveness of various model distillation approaches for the dual-encoder within our proposed framework. We explore four variants of the model distillation methods. The first one, referred to as $\texttt{Model}_{\texttt{Original}}$, represents the model without any distillation. The second one, which is $\texttt{Model}_{\texttt{Single}}$, incorporates only the single modality loss as the distillation method. The third one, which is $\texttt{Model}_{\texttt{Dual}}$, employs solely the dual modality distillation method. The forth one is $\texttt{Model}_{\texttt{SPENCER}}$, which is our proposed distillation approach. And the last one is $\texttt{Model}_{\texttt{SPENCER+Contra}}$, which combines our proposed distillation method with a contrastive loss used in the training of original models.

Table 4 presents the performance comparison results of different pre-trained code retrieval models using these model distillation approaches. Our distillation approach exhibits the best performance across most metrics, confirming the effectiveness of our proposed dual-encoder distillation method. Interestingly, we observe that the performance of the single-modality distillation is generally superior to the dual-modality distillation in most experimental settings. This suggests that the student model effectively learns knowledge from the teacher query encoder by aligning representation vectors to the teacher model, rather than focusing on the relative positional relationship between query and code modalities.

Surprisingly, we find that incorporating the contrastive loss into the model distillation does not contribute to performance improvement; instead, it harms the mode distillation process. The contrastive loss aims to reduce the distance between positive pairs of code and query while increasing the distance between negative pairs, which can be regarded as providing ground-truth labels during training. The reason for its negative impact on our encoder's distillation performance is that the small model has limited ability to construct a good distribution of representation vectors in high-dimensional space. This loss interferes with the learning of the teacher model by the distillation model.

In summary, the distillation with single modality and dual modality has the best performance among all the variants. The introduction of constrastive loss into the model distillation has a negative impact on the distillation performance.

4.3 The influence of the model size to the performance

Table 5 presents the experimental results on the performance of various sizes of distilled query encoders with different pre-trained models. According to the experiment results, the impact of model distillation on precise ranking is observed to be more significant than on rough ranking. Specifically, there is a substantial performance drop in the R@1 metric compared to R@3 and R@5 for models with the same number of layers. The drop in R@5 is only approximately 35% compared to the drop in R@1. These experiment results show that our model distillation method has limited impact on the top K recall ability of the dual-encoder, which indicates that model distillation is feasible for the dual-encoder within our proposed framework.

Moreover, the performance drop for different pre-trained models at the same compression ratio varies. For instance, the performance drop with CodeT5 is much smaller than other pre-trained models while the model is distilled from 12 layers to 9 layers. Furthermore, different pre-trained

TABLE 4
Results of the dual-encoder performance comparison of different pre-trained models with different model distillation approaches. The best results are highlighted in **bold** font.

Model	Python				Java			
	R@1	R@3	R@5	MRR	R@1	R@3	R@5	MRR
CodeBERT$_{\text{Original}}$	0.652	0.839	0.888	0.757	0.533	0.704	0.754	0.633
CodeBERT$_{\text{Single}}$	0.625 (↓4.1%)	0.819 (↓2.4%)	0.876 (↓1.4%)	0.735 (↓2.9%)	0.509 (↓4.5%)	0.687 (↓2.4%)	0.740 (↓1.9%)	0.614 (↓3.0%)
CodeBERT$_{\text{Dual}}$	0.618 (↓5.2%)	0.815 (↓2.9%)	0.872 (↓1.8%)	0.730 (↓3.6%)	0.506 (↓5.1%)	0.686 (↓2.6%)	0.739 (↓2.0%)	0.612 (↓3.3%)
CodeBERT$_{\text{SPENCER}}$	**0.631 (↓3.2%)**	**0.824 (↓1.8%)**	**0.879 (↓1.0%)**	0.740 (↓2.2%)	**0.511 (↓4.1%)**	**0.689 (↓2.1%)**	**0.742 (↓1.6%)**	**0.615 (↓2.8%)**
CodeBERT$_{\text{SPENCER+Contra}}$	**0.631 (↓3.2%)**	0.823 (↓1.9%)	0.878 (↓1.1%)	**0.741 (↓2.1%)**	0.487 (↓8.6%)	0.669 (↓5.0%)	0.727 (↓3.6%)	0.596 (↓5.8%)
GraphCodeBERT$_{\text{Original}}$	0.669	0.853	0.901	0.771	0.541	0.712	0.760	0.640
GraphCodeBERT$_{\text{Single}}$	0.642 (↓4.0%)	0.836 (↓2.0%)	0.889 (↓1.3%)	0.750 (↓2.7%)	0.515 (↓4.8%)	0.692 (↓2.8%)	0.744 (↓2.1%)	0.618 (↓3.4%)
GraphCodeBERT$_{\text{Dual}}$	0.635 (↓5.1%)	0.832 (↓2.5%)	0.886 (↓1.7%)	0.745 (↓3.4%)	0.510 (↓5.7%)	0.688 (↓3.4%)	0.740 (↓2.6%)	0.614 (↓4.1%)
GraphCodeBERT$_{\text{SPENCER}}$	**0.644 (↓3.7%)**	**0.839 (↓1.6%)**	**0.891 (↓1.1%)**	**0.753 (↓2.3%)**	**0.522 (↓3.5%)**	**0.697 (↓2.1%)**	**0.749 (↓1.4%)**	**0.624 (↓2.5%)**
GraphCodeBERT$_{\text{SPENCER+Contra}}$	0.641 (↓4.2%)	0.836 (↓2.0%)	0.889 (↓1.7%)	0.750 (↓2.7%)	0.513 (↓5.2%)	0.690 (↓3.1%)	0.745 (↓2.0%)	0.617 (↓3.6%)
CodeT5$_{\text{Original}}$	0.655	0.842	0.892	0.760	0.500	0.681	0.737	0.608
CodeT5$_{\text{Single}}$	0.632 (↓3.5%)	0.822 (↓2.4%)	0.878 (↓1.6%)	0.741 (↓2.5%)	**0.480 (↓4.0%)**	0.666 (↓2.2%)	0.725 (↓1.6%)	0.590 (↓3.0%)
CodeT5$_{\text{Dual}}$	0.625 (↓4.6%)	0.820 (↓2.6%)	0.877 (↓1.7%)	0.736 (↓3.2%)	0.475 (↓5.0%)	0.661 (↓2.9%)	0.719 (↓2.4%)	0.586 (↓3.6%)
CodeT5$_{\text{SPENCER}}$	**0.639 (↓2.4%)**	**0.828 (↓1.7%)**	**0.882 (↓1.1%)**	**0.746 (↓1.8%)**	**0.480 (↓4.0%)**	**0.667 (↓2.1%)**	**0.726 (↓1.5%)**	**0.591 (↓2.8%)**
CodeT5$_{\text{SPENCER+Contra}}$	0.625 (↓4.6%)	0.821 (↓2.5%)	0.877 (↓1.7%)	0.735 (↓3.3%)	0.469 (↓6.2%)	0.660 (↓3.1%)	0.722 (↓2.0%)	0.583 (↓4.1%)

TABLE 5
Results of the dual-encoder performance comparison of different pre-trained models with different model compression ratio.

Model	Python				Java			
	R@1	R@3	R@5	MRR	R@1	R@3	R@5	MRR
CodeBERT$_{\text{12layers}}$	0.652	0.839	0.888	0.757	0.533	0.704	0.754	0.633
CodeBERT$_{\text{9layers}}$	0.648 (↓0.6%)	0.836 (↓0.4%)	0.886 (↓0.2%)	0.754 (↓0.4%)	0.524 (↓1.7%)	0.697 (↓1.0%)	0.749 (↓0.7%)	0.626 (↓1.1%)
CodeBERT$_{\text{6layers}}$	0.642 (↓1.5%)	0.830 (↓1.1%)	0.882 (↓0.7%)	0.748 (↓1.2%)	0.522 (↓2.1%)	0.696 (↓1.1%)	0.748 (↓0.8%)	0.624 (↓1.4%)
CodeBERT$_{\text{3layers}}$	0.631 (↓3.2%)	0.824 (↓1.8%)	0.879 (↓1.0%)	0.740 (↓2.2%)	0.511 (↓4.1%)	0.689 (↓2.1%)	0.742 (↓1.6%)	0.615 (↓2.8%)
CodeBERT$_{\text{1layer}}$	0.581 (↓10.9%)	0.786 (↓6.3%)	0.848 (↓4.5%)	0.700 (↓7.5%)	0.469 (↓12.0%)	0.652 (↓7.4%)	0.709 (↓6.0%)	0.578 (↓8.7%)
GraphCodeBERT$_{\text{12layers}}$	0.669	0.853	0.901	0.771	0.541	0.712	0.760	0.640
GraphCodeBERT$_{\text{9layers}}$	0.665 (↓0.6%)	0.849 (↓0.5%)	0.898 (↓0.3%)	0.768 (↓0.4%)	0.535 (↓1.1%)	0.708 (↓0.6%)	0.757 (↓0.4%)	0.636 (↓0.6%)
GraphCodeBERT$_{\text{6layers}}$	0.660 (↓1.3%)	0.845 (↓0.9%)	0.896 (↓0.6%)	0.764 (↓0.5%)	0.533 (↓1.5%)	0.706 (↓0.8%)	0.756 (↓0.5%)	0.634 (↓0.9%)
GraphCodeBERT$_{\text{3layers}}$	0.641 (↓4.2%)	0.836 (↓2.0%)	0.889 (↓1.3%)	0.750 (↓2.7%)	0.516 (↓4.6%)	0.691 (↓2.9%)	0.743 (↓2.2%)	0.619 (↓3.3%)
GraphCodeBERT$_{\text{1layer}}$	0.607 (↓9.3%)	0.810 (↓5.0%)	0.867 (↓3.8%)	0.722 (↓6.4%)	0.483 (↓10.7%)	0.662 (↓7.0%)	0.719 (↓5.4%)	0.589 (↓8.0%)
CodeT5$_{\text{12layers}}$	0.655	0.842	0.892	0.760	0.500	0.681	0.737	0.608
CodeT5$_{\text{9layers}}$	0.654 (↓0.2%)	0.840 (↓0.2%)	0.890 (↓0.2%)	0.758 (↓0.3%)	0.497 (↓0.6%)	0.678 (↓0.4%)	0.735 (↓0.3%)	0.606 (↓0.3%)
CodeT5$_{\text{6layers}}$	0.650 (↓0.8%)	0.835 (↓0.8%)	0.888 (↓0.4%)	0.756 (↓0.5%)	0.491 (↓1.8%)	0.673 (↓1.2%)	0.733 (↓0.5%)	0.600 (↓1.3%)
CodeT5$_{\text{3layers}}$	0.639 (↓2.4%)	0.828 (↓1.7%)	0.882 (↓1.1%)	0.746 (↓1.8%)	0.480 (↓4.0%)	0.667 (↓2.1%)	0.726 (↓1.5%)	0.591 (↓2.8%)
CodeT5$_{\text{1layer}}$	0.597 (↓8.9%)	0.800 (↓5.0%)	0.858 (↓3.8%)	0.713 (↓6.2%)	0.466 (↓6.8%)	0.654 (↓4.0%)	0.717 (↓2.7%)	0.578 (↓4.9%)

models demonstrate distinct performance drop trends with an increasing model compression ratio. For most distilled models, the performance drop accelerates when distilling to 3 layers and becomes considerably larger at 1 layer. However, the performance drop of CodeT5 increases at a slower rate compared to other pre-trained models as its compression ratio increases.

Finally, it's worth noting that even for the same distilled model, the performance varies across different datasets. Specifically, we can find that the performance drop of CodeBERT which is distilled to 3 layers on the Python dataset is smaller than the performance drop of it on the Java dataset, and the experiment results are opposite for the rest of the distilled pre-trained models.

In summary, the extent of performance degradation during model distillation varies greatly based on the choice of mode compression ratio, the pre-trained models, and the datasets.

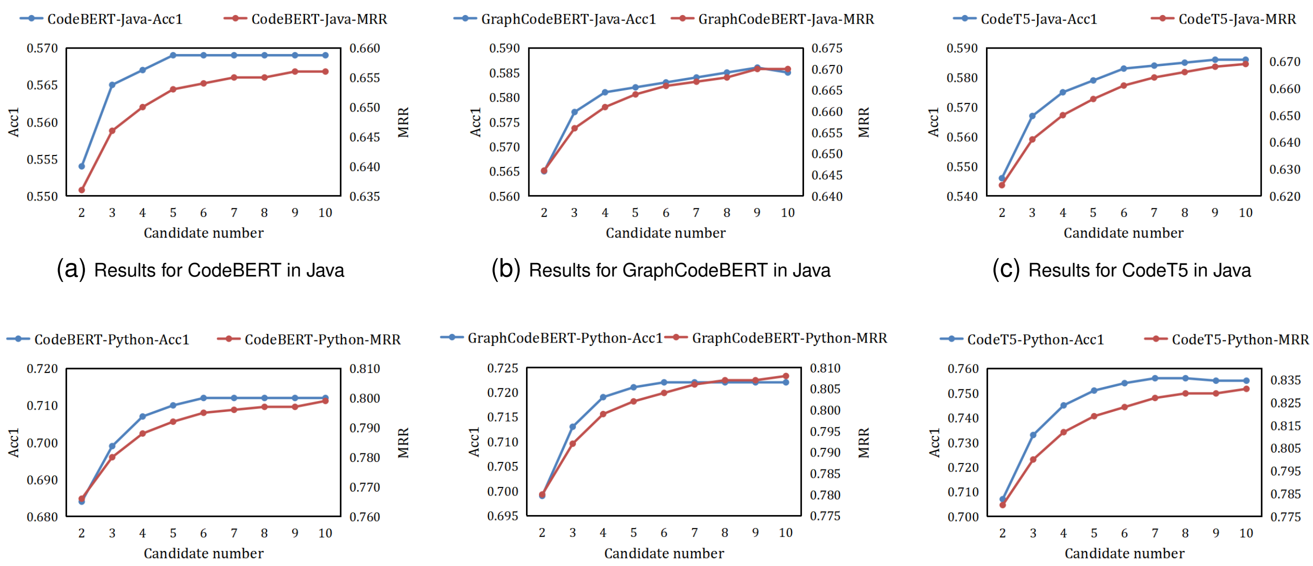
4.4 The impact of different training strategy to the performance with the same model size

Table 6 presents the experiment results for evaluating the performance of the dual encoder under different training strategies. The four models compared are Model$_{\text{Original}}$, which represents the original query model with 12 layers trained with the original code encoder; Model$_{\text{DirectTrain}}$, denoting the query model with 3 layers directly trained with the original code encoder; Model$_{\text{DirectDistill}}$, representing the query encoder with 3 layers directly distilled from the original query encoder; and Model$_{\text{SPENCER}}$, which is the query encoder distilled from the original query encoder using our proposed strategy.

Based on the experiment results, we observe that both Model$_{\text{DirectDistill}}$ and Model$_{\text{SPENCER}}$ outperform Model$_{\text{DirectTrain}}$ across all metrics and pre-trained models. This demonstrates the effectiveness of the model distillation. Additionally, our proposed distillation strategy shows the capability to further enhance the performance of some pre-trained models compared to the strategy of direct distil-

TABLE 6
Results of distillated dual-encoder performance comparison of different pre-trained models with different training strategy. The best results are highlighted in bold font.

Model	Python R@1	Python R@3	Python R@5	Python MRR	Java R@1	Java R@3	Java R@5	Java MRR
CodeBERT$_{Original}$	0.652	0.839	0.888	0.757	0.533	0.704	0.754	0.633
CodeBERT$_{DirectTrain}$	0.591 (↓9.4%)	0.799 (↓4.7%)	0.851 (↓4.2%)	0.706 (↓6.7%)	0.458 (↓14.1%)	0.659 (↓6.4%)	0.705 (↓6.5%)	0.569 (↓10.1%)
CodeBERT$_{DirectDistill}$	**0.631 (↓3.2%)**	**0.824 (↓1.8%)**	**0.879 (↓1.0%)**	**0.740 (↓2.2%)**	**0.511 (↓4.1%)**	**0.689 (↓2.1%)**	**0.742 (↓1.6%)**	**0.615 (↓2.8%)**
CodeBERT$_{SPENCER}$	**0.631 (↓3.2%)**	**0.824 (↓1.8%)**	**0.879 (↓1.0%)**	**0.740 (↓2.2%)**	**0.511 (↓4.1%)**	**0.689 (↓2.1%)**	**0.742 (↓1.6%)**	**0.615 (↓2.8%)**
GraphCodeBERT$_{Original}$	0.669	0.853	0.901	0.771	0.541	0.712	0.760	0.640
GraphCodeBERT$_{DirectTrain}$	0.609 (↓9.0%)	0.809 (↓5.2%)	0.866 (↓3.9%)	0.722 (↓6.4%)	0.471 (↓12.9%)	0.659 (↓7.4%)	0.718 (↓5.5%)	0.583 (↓8.9%)
GraphCodeBERT$_{DirectDistill}$	0.641 (↓4.2%)	0.836 (↓2.0%)	0.889 (↓1.3%)	0.750 (↓2.7%)	0.516 (↓4.6%)	0.691 (↓2.9%)	0.743 (↓2.2%)	0.619 (↓3.3%)
GraphCodeBERT$_{SPENCER}$	**0.644 (↓3.7%)**	**0.839 (↓1.6%)**	**0.891 (↓1.1%)**	**0.753 (↓2.3%)**	**0.522 (↓3.5%)**	**0.697 (↓2.1%)**	**0.749 (↓1.4%)**	**0.624 (↓2.5%)**
CodeT5$_{Original}$	0.655	0.842	0.892	0.760	0.500	0.681	0.737	0.608
CodeT5$_{DirectTrain}$	0.622 (↓5.0%)	0.817 (↓3.0%)	0.872 (↓2.2%)	0.732 (↓3.7%)	0.446 (↓10.8%)	0.633 (↓7.0%)	0.696 (↓5.6%)	0.559 (↓8.1%)
CodeT5$_{DirectDistill}$	**0.639 (↓2.4%)**	**0.828 (↓1.7%)**	**0.882 (↓1.1%)**	**0.746 (↓1.8%)**	**0.480 (↓4.0%)**	**0.667 (↓2.1%)**	**0.726 (↓1.5%)**	**0.591 (↓2.8%)**
CodeT5$_{SPENCER}$	**0.639 (↓2.4%)**	**0.828 (↓1.7%)**	**0.882 (↓1.1%)**	**0.746 (↓1.8%)**	**0.480 (↓4.0%)**	**0.667 (↓2.1%)**	**0.726 (↓1.5%)**	**0.591 (↓2.8%)**



(a) Results for CodeBERT in Java
(b) Results for GraphCodeBERT in Java
(c) Results for CodeT5 in Java
(d) Results for CodeBERT in Python
(e) Results for GraphCodeBERT in Python
(f) Results for CodeT5 in Python

Fig. 4. Overall performance comparison between SPENCER with different number of recall candidates based on different pre-trained models

lation. Specifically, the best performance is achieved with GraphCodeBERT on both Python and Java datasets using our distillation strategy. Interestingly, the distillation strategy has no impact on the pre-trained models named CodeBERT and CodeT5, indicating that involving a teaching assistant in the model distillation process is unnecessary for these two models. These results suggest that the necessity of the teaching assistant during model distillation depends on the model architecture of the pre-trained models.

In conclusion, our proposed distillation strategy can outperform both direct training of a small model and direct distillation strategy. Moreover, the selection of a teaching assistant model depends on the specific pre-trained models, as not all of them require a teaching assistant during the distillation process. This highlights the effectiveness and adaptability of our approach, demonstrating its potential to achieve superior performance.

4.5 The impact of the recall number of the code candidates to the overall performance of SPENCER

Fig. 4 displays the experiment results about the impact of the recall number of candidates on the overall performance of our proposed SPENCER across various pre-trained models. The experimental results indicate that the overall performance increase of our SPENCER varies across different pre-trained models as the recall number increases. Specifically, we observe a significant boost in our SPENCER's overall performance when the recall number is increased from 2 to 5 for the CodeBERT and GraphCodeBERT. This performance increase tends to stabilize beyond a recall number of 5. However, the performance improvement continues with increasing recall numbers for the pre-trained model named CodeT5. Furthermore, it's worth noting that the impact of recall number on the overall performance of SPENCER on the MRR metric is more substantial compared to the R@1 metric. While R@1 exhibits only marginal growth as the candidate number exceeds 5, the overall performance on MRR continues to be improved with higher recall numbers. These results indicate that although sometimes the dual-encoder fails to return the precise code snippet that the cross-encoder ranks as the top 1 answer, it does have the capability to retrieve accurate code snippets that can be ranked as sub-optimal answers by the cross-encoder when the recalled candidates number from dual-encoder increases.

In conclusion, our proposed framework's overall performance exhibits steady improvement as the number of recall candidates from the dual-encoder increases. Nevertheless, the extent of this performance improvement depends on the pre-trained models we have adopted in our framework. Furthermore, it is noteworthy that the increase in the number of recalls has a more pronounced effect on the overall performance of SPENCER on the MRR metric compared to the R@1 metric.

5 Threats to Validity

After careful analysis, we have identified several potential threats to the validity of our study.

5.1 Threats to External Validity

We have chosen Python and Java datasets to evaluate the efficiency of our proposed framework, taking into account training costs. Nonetheless, it is essential to acknowledge that the performance of our framework might vary across different programming languages.

Furthermore, we deliberately limit our choice to three pre-trained models in our proposed framework, taking into account the constraints of experimental costs. It is possible that the performance improvement of our proposed framework is not so significant or the distillation approaches inside our framework will have a higher performance loss when we adopt other pre-trained models as the base model in our framework.

Finally, we assess the presented approach solely utilizing the accuracy and Mean Reciprocal Rank (MRR) metrics in the comprehensive performance experiment. Nevertheless, it's important to note that the overall efficacy of our proposed framework might exhibit variations when considered through different metrics.

5.2 Threats to Internal Validity

In this study, we maintain consistency by utilizing the identical hyperparameters as CodeBERT for all the pre-trained models. While we acknowledge that variations in hyperparameters could potentially affect overall model performance, we refrained from exploring such influences due to the high costs associated with fine-tuning the models. For example, the training batch size is a very important hyperparameter for the dual-encoder training, since we adopt the contrastive loss to train the dual-encoder and previous research shows that the increase of training batch size can improve the performance. Nevertheless, we have omitted an exploration of the impact of the training batch size on the dual-encoder's behavior.

6 Related Work

6.1 Code Retrieval

In this subsection, we briefly introduce the deep learning-based code retrieval approaches, which are classified into non pre-training based approaches and pre-training based approaches.

6.1.1 Non pre-training approaches

Sachdev et al. [16] carry out the techniques on natural language processing directly to the code area and investigate the performance of techniques including wording embedding [17], TF-IDF [18] weighting, and high-dimensional vector similarity search [19] in the task of code retrieval. Cambronero et al. [20] evaluate the performance of supervised and unsupervised techniques in the neural networks and demonstrate the effectiveness of the supervised training in the code retrieval task. Gu et al. [5] extract the code tokens, method name tokens, and API sequences from the original code at first. These features will be embedded into the feature vectors individually and finally fused into a single representation vectors for the given code. Husain et al. [21] construct an open-source dataset for the code retrieval and find that the self-attention model achieves the best performance among all the models through their evaluation. Yao et al. [22] adopt reinforcement learning to generate the code annotation at first and such code annotation can help the code retrieval model to better distinguish the relevant code snippets from other similar code. Gu et al. [4] extract the program dependency graph from the given code and convert the graph into the relationship matrix. The generated matrix will be concatenated with the statement-level representation vectors and fed into long short-term memory (LSTM) networks to generate function-level representation vector.

6.1.2 Pre-training approaches

Inspired by the pre-training models in natural language processing, Feng et al. [9] proposed a bimodal pre-trained model with Transformer-based neural architecture, which is named CodeBERT. CodeBERT is trained with the pre-training task of replaced token detection. Later, Guo et al. [7] considered the inherent structure of code and proposed a pre-trained model named GraphCodeBERT. GraphCodeBERT is trained with the extra information of data flow. To address the problem that previous pre-training models are sensitive to the source code edits, Jain et al. [23] pre-trained ContrCode to identify the functionally similar variants among non-equivalent distractors. Ahmad [24] proposed a sequence to sequence pre-trained which trained via denoising autoencoding. Unlike previous pre-training models which only contain the encoder, Wang et al. [25] proposed a unified pre-trained encoder-decoder Transformer model named CodeT5. CodeT5 is trained with the identifier-aware pre-training task and such a task enables the model to distinguish the code tokens belonging to identifiers and recover the masked identifiers. Similarly, Niu et al. [26] proposed SPT-Code with three pre-training tasks which enable SPT-Code to learn knowledge of source code, the corresponding code structure, and a natural language description of the code without relying on any bilingual corpus. To further involve symbolic and syntactic properties of source code into the pre-training model, Wang et al. [27] proposed SyncoBERT trained with two novel pre-training objectives which are Identifier Prediction and AST Edge Prediction. To address the problem that the encoder-decoder framework is sub-optimal for auto-regressive tasks, Guo et al. [6] proposed a unified cross-modal pre-trained model named UniXcoder. To control the behavior of the model, UniXcoder utilizes mask attention matrices with prefix adapters. Bui et al. [28] proposed a self-supervised contrastive learning framework named Corder, which can learn to distinguish similar and dissimilar code snippets.

6.2 Knowledge Distillation

The technology of knowledge distillation aims to reduce the model parameters while preserving most of the performance of the original model by making the small model learn the output distribution from the large model. Such technology has attracted a large number of researchers in recent years. Hinton et al. [29] first proposed the concept of knowledge distillation. Li et al. proposed a mimic method that can map the features from the small network onto the same dimension of the large network for knowledge distillation. Tang et al. [30] distillated a Bi-LSTM model from BERT [31] for the task of paraphrasing, natural language inference, and sentiment classification. Romero et al. [32] adopted a deeper and thinner student network to learn the knowledge from the teacher network and achieve a better performance with fewer parameters on CIFAR-10. To further improve the efficiency of search model in the recommendation system, Tang et al. [33] proposed a knowledge distillation technique to train a student model by learning the ranking knowledge of documents/items from both the training data and teacher model. The student model can achieve a comparable performance as the teacher model with a more efficient online inference time. Zhang et al. [34] proposed a deep mutual learning (DML) strategy which makes the multiple student models to learn collaboratively and teach each other during the training process. The experiment results show that the mutual learning of many student models outperforms distillation from a teacher model. Rather than training a smaller student model from the large teacher model, Tommaso et al. [35] trained student models which are parameterized identically to the teachers models and they found that the student models outperform their teachers significantly on both computer vision and language modeling tasks. To avoid the full training of a large model, Li et al. [36] proposed a online knowledge distillation approach that acquires the predicted heatmaps from the trained multi-branch network and assemble these heatmaps as the target heatmaps to teach each branch in reverse. Most of the previous research on Knowledge distillation has primarily concentrated on classification tasks. However, its potential application in tasks involving the generation of representation vectors, such as the code retrieval task, remains relatively unexplored.

7 Conclusion

In this paper, we introduce a framework that seamlessly integrates both dual-encoder and cross-encoder for code retrieval tasks. Additionally, we present an innovative approach to distill the query encoder model which can improve the inference efficiency of the query encoder while preserving most of its performance. To further elevate the performance of these distilled models while maintaining consistent model sizes, we propose a novel teaching assistant selection strategy for the distillation process. Our experimental results show the effectiveness of our proposed framework. Notably, our model distillation approach succeeds in reducing the inference time of the query encoder within our framework by approximately 70% while preserving over 98% of the overall performance.

In the future, our focus will be on investigating methods to further reduce the inference time of the query encoder while enhancing the overall performance of this framework.

References

[1] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, "Two studies of opportunistic programming: interleaving web foraging, learning, and writing code," in *Proceedings of the 27th International Conference on Human Factors in Computing Systems, CHI 2009, Boston, MA, USA, April 4-9, 2009*, D. R. O. Jr., R. B. Arthur, K. Hinckley, M. R. Morris, S. E. Hudson, and S. Greenberg, Eds. ACM, 2009, pp. 1589–1598. [Online]. Available: https://doi.org/10.1145/1518701.1518944

[2] F. Lv, H. Zhang, J. Lou, S. Wang, D. Zhang, and J. Zhao, "Codehow: Effective code search based on API understanding and extended boolean model (E)," in *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, M. B. Cohen, L. Grunske, and M. Whalen, Eds. IEEE Computer Society, 2015, pp. 260–270. [Online]. Available: https://doi.org/10.1109/ASE.2015.42

[3] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: finding relevant functions and their usage," in *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*, R. N. Taylor, H. C. Gall, and N. Medvidovic, Eds. ACM, 2011, pp. 111–120. [Online]. Available: https://doi.org/10.1145/1985793.1985809

[4] W. Gu, Z. Li, C. Gao, C. Wang, H. Zhang, Z. Xu, and M. R. Lyu, "Cradle: Deep code retrieval based on semantic dependency learning," *Neural Networks*, vol. 141, pp. 385–394, 2021. [Online]. Available: https://doi.org/10.1016/j.neunet.2021.04.019

[5] 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. [Online]. Available: https://doi.org/10.1145/3180155.3180167

[6] 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. [Online]. Available: https://doi.org/10.18653/v1/2022.acl-long.499

[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. [Online]. Available: https://openreview.net/forum?id=jLoC4ez43PZ

[8] O. Khattab and M. Zaharia, "Colbert: Efficient and effective passage search via contextualized late interaction over BERT," in *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, J. X. Huang, Y. Chang, X. Cheng, J. Kamps, V. Murdock, J. Wen, and Y. Liu, Eds. ACM, 2020, pp. 39–48. [Online]. Available: https://doi.org/10.1145/3397271.3401075

[9] 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. [Online]. Available: https://doi.org/10.18653/v1/2020.findings-emnlp.139

[10] T. Gao, X. Yao, and D. Chen, "Simcse: Simple contrastive learning of sentence embeddings," 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*, M. Moens, X. Huang, L. Specia, and S. W. Yih, Eds. Association for Computational Linguistics, 2021, pp. 6894–6910. [Online]. Available: https://doi.org/10.18653/v1/2021.emnlp-main.552

[11] R. Haldar, L. Wu, J. Xiong, and J. Hockenmaier, "A multi-perspective architecture for semantic code search," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, D. Jurafsky, J. Chai, N. Schluter, and J. R. Tetreault, Eds. Association for Computational Linguistics, 2020, pp. 8563–8568. [Online]. Available: https://doi.org/10.18653/v1/2020.acl-main.758

[12] J. Shuai, L. Xu, C. Liu, M. Yan, X. Xia, and Y. Lei, "Improving code search with co-attentive representation learning," in *ICPC '20: 28th International Conference on Program Comprehension, Seoul, Republic of Korea, July 13-15, 2020*. ACM, 2020, pp. 196–207. [Online]. Available: https://doi.org/10.1145/3387904.3389269

[13] S. Fang, Y. Tan, T. Zhang, and Y. Liu, "Self-attention networks for code search," *Inf. Softw. Technol.*, vol. 134, p. 106542, 2021. [Online]. Available: https://doi.org/10.1016/j.infsof.2021.106542

[14] G. Heyman and T. V. Cutsem, "Neural code search revisited: Enhancing code snippet retrieval through natural language intent," *CoRR*, vol. abs/2008.12193, 2020. [Online]. Available: https://arxiv.org/abs/2008.12193

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[16] 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. [Online]. Available: https://doi.org/10.1145/3211346.3211353

[17] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Trans. Assoc. Comput. Linguistics*, vol. 5, pp. 135–146, 2017. [Online]. Available: https://doi.org/10.1162/tacl_a_00051

[18] B. Mitra and N. Craswell, "Neural models for information retrieval," *CoRR*, vol. abs/1705.01509, 2017. [Online]. Available: http://arxiv.org/abs/1705.01509

[19] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *IEEE Trans. Big Data*, vol. 7, no. 3, pp. 535–547, 2021. [Online]. Available: https://doi.org/10.1109/TBDATA.2019.2921572

[20] 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. [Online]. Available: https://doi.org/10.1145/3338906.3340458

[21] 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. [Online]. Available: http://arxiv.org/abs/1909.09436

[22] Z. Yao, J. R. Peddamail, and H. Sun, "Coacor: Code annotation for code retrieval with reinforcement learning," in *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, L. Liu, R. W. White, A. Mantrach, F. Silvestri, J. J. McAuley, R. Baeza-Yates, and L. Zia, Eds. ACM, 2019, pp. 2203–2214. [Online]. Available: https://doi.org/10.1145/3308558.3313632

[23] P. Jain, A. Jain, T. Zhang, P. Abbeel, J. Gonzalez, and I. Stoica, "Contrastive code representation learning," 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*, M. Moens, X. Huang, L. Specia, and S. W. Yih, Eds. Association for Computational Linguistics, 2021, pp. 5954–5971. [Online]. Available: https://doi.org/10.18653/v1/2021.emnlp-main.482

[24] 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. [Online]. Available: https://doi.org/10.18653/v1/2021.naacl-main.211

[25] Y. Wang, W. Wang, S. R. Joty, and S. C. H. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, M. Moens, X. Huang, L. Specia, and S. W. Yih, Eds. Association for Computational Linguistics, 2021, pp. 8696–8708. [Online]. Available: https://doi.org/10.18653/v1/2021.emnlp-main.685

[26] 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. [Online]. Available: https://doi.org/10.1145/3510003.3510096

[27] X. Wang, Y. Wang, F. Mi, P. Zhou, Y. Wan, X. Liu, L. Li, H. Wu, J. Liu, and X. Jiang, "Syncobert: Syntax-guided multi-modal contrastive pre-training for code representation," *arXiv preprint arXiv:2108.04556*, 2021.

[28] 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. [Online]. Available: https://doi.org/10.1145/3404835.3462840

[29] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *CoRR*, vol. abs/1503.02531, 2015. [Online]. Available: http://arxiv.org/abs/1503.02531

[30] R. Tang, Y. Lu, L. Liu, L. Mou, O. Vechtomova, and J. Lin, "Distilling task-specific knowledge from BERT into simple neural networks," *CoRR*, vol. abs/1903.12136, 2019. [Online]. Available: http://arxiv.org/abs/1903.12136

[31] 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. [Online]. Available: https://doi.org/10.18653/v1/n19-1423

[32] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6550

[33] J. Tang and K. Wang, "Ranking distillation: Learning compact ranking models with high performance for recommender system," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, Y. Guo and F. Farooq, Eds. ACM, 2018, pp. 2289–2298. [Online]. Available: https://doi.org/10.1145/3219819.3220021

[34] Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu, "Deep mutual learning," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 2018, pp. 4320–4328. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2018/html/Zhang_Deep_Mutual_Learning_CVPR_2018_paper.html

[35] T. Furlanello, Z. C. Lipton, M. Tschannen, L. Itti, and A. Anandkumar, "Born-again neural networks," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018, pp. 1602–1611. [Online]. Available: http://proceedings.mlr.press/v80/furlanello18a.html

[36] Z. Li, J. Ye, M. Song, Y. Huang, and Z. Pan, "Online knowledge distillation for efficient pose estimation," in *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, 2021, pp. 11 720–11 730. [Online]. Available: https://doi.org/10.1109/ICCV48922.2021.01153