CONSIDER: Commonalities and Specialties Driven Multilingual Code Retrieval Framework

Rui Li[1, 2], **Liyang He**[1, 2], **Qi Liu**[1, 2*], **Yuze Zhao**[1, 2], **Zheng Zhang**[1, 2], **Zhenya Huang**[1, 2], **Yu Su**[3], **Shijin Wang**[2, 4]

[1]Anhui Province Key Laboratory of Big Data Analysis and Application & School of Computer Science and Technology, University of Science and Technology of China
[2]State Key Laboratory of Cognitive Intelligence
[3]School of Computer Science and Artificial Intelligence, Hefei Normal University
[4]iFLYTEK AI Research (Central China), iFLYTEK Co., Ltd.
{ruili2000, heliyang, yuzezhao, zhangzheng}@mail.ustc.edu.cn, {huangzhy, qiliuql}@ustc.edu.cn, yusu@hfnu.edu.cn, sjwang3@iflytek.com

Abstract

Multilingual code retrieval aims to find code snippets relevant to a user's query from a multilingual codebase, which plays a crucial role in software development and expands their application scenarios compared to classical monolingual code retrieval. Despite the performance improvements achieved by previous studies, two crucial problems are overlooked in the multilingual scenario. First, certain programming languages face data scarcity in specific domains, resulting in limited representation capabilities within those domains. Second, different programming languages can be used interchangeably within the same domain, making it challenging for multilingual models to accurately identify the intended programming language of a user's query. To address these issues, we propose the **C**omm**ON**alities and **S**pec**I**alties **D**riven Multilingual Cod**E** **R**etrieval Framework (CONSIDER), which includes two modules. The first module enhances the representation of various programming languages by modeling pairwise and global commonalities among them. The second module introduces a novel contrastive learning negative sampling algorithm that leverages language confusion to automatically extract specific language features. Through our experiments, we confirm the significant benefits of our model in real-world multilingual code retrieval scenarios in various aspects. Furthermore, an evaluation demonstrates the effectiveness of our proposed CONSIDER framework in monolingual scenarios as well. Our source code is available at https://github.com/smsquirrel/consider.

Introduction

Code retrieval is a foundational task in code intelligence (Mukherjee, Jermaine, and Chaudhuri 2020; Kim et al. 2010). As illustrated in Figure 1 (a), given a natural language query and a selected programming language, the classical monolingual code retrieval model aims to find code snippets in a large-scale codebase (Haldar et al. 2020; Wan et al. 2019). This system can assist developers in code reuse (Shuai et al. 2020; Nie et al. 2016) and understanding the complex software libraries (Ling et al. 2021). With the advancement of artificial intelligence technologies, many advanced monolingual code retrieval methods (Cambronero et al. 2019; Chen and Zhou 2018; Shuai et al. 2020) have been proposed and made tremendous progress.

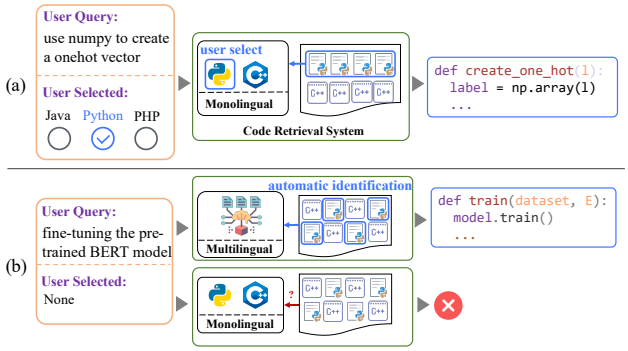


Figure 1: (a) Monolingual scenarios: the user's query and selected language guide the system's choice of model and code repository. (b) Multilingual scenarios: the retrieval model autonomously searches for code snippets in a multilingual library without user-specified language selection.

As the scope of application scenarios expands, there has been a surge in the demand for code retrieval. For example, software projects hosted on code repositories like GitHub[1] and GitLab[2] are increasingly developed using multiple programming languages. This trend has led to a growing need for multilingual code retrieval capabilities (Li, Xu, and Chen 2022; Ma et al. 2023). Compared to monolingual code retrieval, multilingual code retrieval models provide significant advantages. First, they eliminate the requirement for deploying and maintaining multiple retrieval models for different languages, resulting in cost reduction. Second, as illustrated in Figure 1 (b), multilingual models compare the similarity between queries and various programming languages, thereby extending the scope of retrieval scenarios. This capability allows them to retrieve the relevant code

*Corresponding Author.

[1]https://github.com
[2]https://about.gitlab.com/

from a multilingual codebase. However, existing work faces the following two problems in the scenario of multilingual code retrieval.

First, some programming languages often encounter the problem of data scarcity in certain specific domains. For example, in the domain of deep learning, the C++ language plays a significant role due to its exceptional performance. However, due to the complexity of C++ and its steep learning curve, there are fewer available resources in this domain. In contrast, Python offers a plethora of libraries and frameworks for deep learning, making it rich in application resources within this domain. In such cases, we can leverage the rich information provided by Python in deep learning to enhance the representation of C++ code in this domain. Therefore, it is necessary to exploit the commonalities of programming languages in multilingual code retrieval models to enhance performance in the scenario where data is scarce for certain programming languages.

Second, while the commonalities among programming languages can be beneficial in data-scarce scenarios, they can also pose challenges for current multilingual code retrieval models in accurately determining the intended language for a given query. Considering an example query in Figure 1 (b): "fine-tuning the pre-trained BERT model." In this case, the retrieval model may struggle to discern whether the intended language is C++ or Python, as both languages have applications in the field of deep learning. However, it is worth noting that C++ is often employed for hardware acceleration and deployment optimization in deep learning, whereas Python is commonly chosen for model design and training processes. Therefore, the target language is more likely to be Python in this context. Consequently, it is crucial to incorporate the specialties of programming languages into the modeling process to accurately identify the language intent of a user's query in multilingual scenarios.

To this end, we propose the **C**omm**ON**alities and **S**pec**I**alties **D**riven Multilingual Cod**E** **R**etrieval Framework (CONSIDER) to tackle aforementioned problems, which facilitates the seamless integration of existing pre-trained Transformer models into multilingual scenarios. This model consists of two modules. First, to capture the commonalities between programming languages, we introduce the paired commonality extraction module and the global commonality extraction module. These modules enable us to effectively model the commonalities and enhance the code representation across different languages. Second, in order to model the specificity of programming languages, we propose a novel algorithm for sampling negative examples in contrastive learning. We utilize confusion matrices between different programming languages to construct negative samples, aiming to automatically capture the differences in easily confused languages. Besides, introducing a confusion matrix during the sampling process leads to an imbalance within and between languages. To address this issue, we further employ a balancing distribution technique to ensure stable training. By combining these modules, CONSIDER offers a comprehensive approach to address the challenges of accurately identifying language intent and enhancing performance in data-scarce scenarios within multilingual code retrieval models. In summary, our main contributions can be summarized as follows:

- We propose a novel framework CONSIDER for multilingual code retrieval that incorporates two crucial aspects of multiple programming languages: their commonalities and their specialties.
- We introduce a pairwise commonality extraction module and a global commonality extraction module to model the commonalities of programming languages, enabling effective modeling of the shared characteristics among programming languages.
- We propose the Confusion-Matrix-Guided Sampling Algorithm, which leverages confusion matrices to capture the specialties of programming languages, thereby enhancing the ability of discerning the query intend.
- We conduct experiments in real-world multilingual retrieval scenarios, demonstrating the unique advantages of our proposed CONSIDER framework compared to other multilingual code retrieval models in this scenario. The experimental results also prove that our model can enhance the performance of multiple languages in monolingual scenarios.

Related Work

Code Retrieval. We mainly introduce code retrieval in two parts: monolingual code retrieval models and multilingual code retrieval models. For monolingual code retrieval models, one type of research work involves query enhancement (Arakelyan et al. 2022; Lv et al. 2015; Lemos et al. 2014; Zhang et al. 2018). This method aims to supplement the query with additional knowledge, increasing the information of the query before matching it with the code. A second method involves multi-perspective modeling of the code (Chen and Zhou 2018; Kim et al. 2010; Zubkov et al. 2022), using technologies such as AST, CFG, DFG to extract structured features of the code and thereby strengthen the representation of the code. Another approach comes from the perspective of multitask learning (Yao, Peddamail, and Sun 2019; Ye et al. 2020), enhancing the retrieval task through the design of auxiliary tasks related to retrieval, including tasks of annotation generation and code generation.

As for multilingual code retrieval models, there is presently less related work. One method employs knowledge distillation for multilingual code retrieval; the idea is to first train a monolingual teacher model, and then use this monolingual teacher model to guide the training of a multilingual student model. This method effectively trains a multilingual code retrieval model. The second approach involves using the LLVM compiler to pre-generate a consistent intermediate representation (IR) for each programming language. This method can obtain a unified representation across languages. However, both of these methods overlook modeling the commonalities among programming languages, leading to suboptimal performance in scenarios with sparse programming language data. Additionally, they struggle to identify language intent from the linguistic features within user queries.

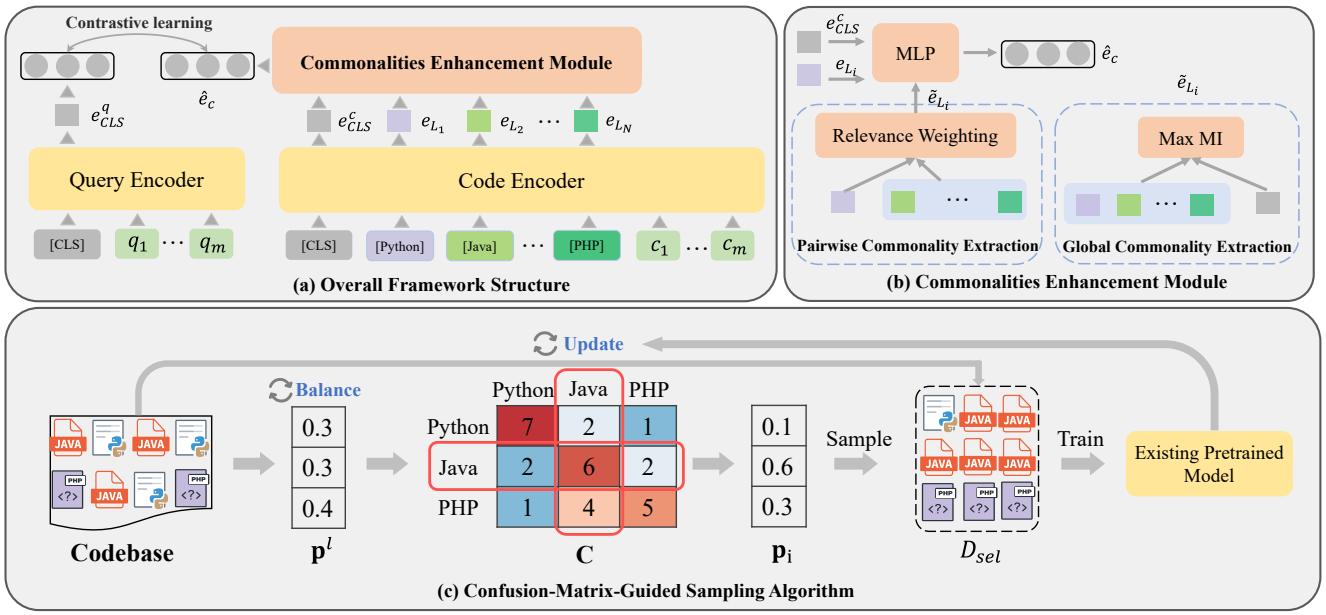


Figure 2: The framework of CONSIDER.

Contrastive learning. Contrastive learning is a technique that aims to make representations agree with each other under proper transformations. It has attracted the attention of researchers across all fields, including CV (He et al. 2020), NLP (He et al. 2023; Giorgi et al. 2021; Zhao et al. 2023), and other domains (Tong et al. 2020; Ning et al. 2023).

Recently, researchers in code retrieval have also begun leveraging contrastive learning techniques to enhance task performance. (Bui, Yu, and Jiang 2021) primarily employs semantically preserving program transformations to generate functionally equivalent code snippets as positive samples for contrastive learning, aiming to identify semantically equivalent and non-equivalent code segments. (Li et al. 2022) and colleagues construct positive contrastive learning samples through representation-level data augmentation. (Huang et al. 2021) introduced CoCLR, which uses query rewriting techniques like random word deletion as positive samples for query in contrastive learning.

Differing from prior researches, we have devised a novel contrastive learning approach specifically tailored for multilingual code retrieval, aiming to model programming language characteristics. Compared to other contrastive learning negative sample construction methods, our method essentially is a heuristic approach for constructing negative samples, which does not require additional inference overhead for the selection of negative samples.

CONSIDER Framework

Problem Definition

Code Retrieval. Given a (query, code snippet) space (Q, C). We denote a pair of (query, code snippet) as $(q, c) \in (Q, C)$, where $q = \{q_1, q_2, ..., q_n\}$ is a query composed of n tokens, and $c = \{c_1, c_2, ..., c_m\}$ is a sequence of code snippets composed of m tokens. Our goal is to train a model f, we can find the code snippet c with the highest matching score according to f:

$$\forall q \in Q, \max_{c \in C} f(q, c), \tag{1}$$

where $f(q, c)$ represents the matching score between the query q and the code snippet c.

This formalization can be adapted to address both monolingual and multilingual code retrieval tasks. In the monolingual case, the (Q, C) space is composed of one programming language dataset. In contrast, multilingual code retrieval involves a (query, code snippet) space (Q, C) that encompasses multiple language datasets, denoted as $Q = \cup_{i=1}^{N} Q_i$ and $C = \cup_{i=1}^{N} C_i$. Q_i and C_i represent queries and code snippets from the (query, code snippet) space of language i, respectively.

Framework Overview

We design two structures to address the two challenges mentioned above. Figure 2 depicts our framework. First, we model different attention patterns for different languages by adding language tokens, which allows for better representation modeling of different languages; Second, to extract language commonalities, we design paired commonality extraction modules and global commonality extraction modules, which are used to extract the commonalities among all languages and between language pairs, respectively; Third, for modeling language-specific features, based on contrastive learning, we introduce a novel negative sampling algorithm. By utilizing the confusion matrix of multiple languages on the validation set, we sample the languages that are easily confused with the selected base language as negative samples, thus automatically learning language-specific features.

Feature Extraction

We begin with a bi-encoder architecture that utilizes pre-trained Transformer models as the backbone neural network, as depicted in Figure 2(a). To capture the representation of the query, we prepend a $[CLS]$ token before the original query's token q, resulting in q':

$$q' = [CLS] \circ q \circ [SEP], \tag{2}$$

where the $\circ$ represents concatenation operation and $[SEP]$ is a special token used to denote the end of input. Subsequently, q' is input into the query encoder E_ϕ, yielding the output $E_\phi(q')$, where ϕ denotes the parameters of the query encoder. The representation of the $[CLS]$ token, denoted as e^q_{CLS}, serves as the representation of the query q.

To effectively model programming language features, we introduce language tokens and employ distinct attention mechanisms to extract features from different programming languages. For each programming language i, we introduce a language token denoted as $[L_i]$. We append all the language tokens to the code snippet c and add a $[CLS]$ token at the beginning, resulting in a new input c':

$$c' = [CLS] \circ [L_1] \ldots [L_n] \circ c \circ [SEP]. \tag{3}$$

To ensure seamless integration of the introduced $[L]$ tokens into the model, we initialize them using the embedding of the $[CLS]$ token, because the $[CLS]$ token tends to capture the overall meaning of the entire sentence (Clark et al. 2019; Kovaleva et al. 2019). Additionally, to prevent any interference with the positional encoding of the original input sentences, we set the position ids of all $[L_i]$ tokens to 0, while the original code segments are marked starting from 1. Then, we feed c' into the code encoder E_θ to get the output $E_\theta(c')$, where ϕ denotes the parameters of the code encoder. The representations of the $[CLS]$ tokens are denoted as e^c_{CLS}, and the representations of the $[L]$ tokens are denoted as e_{L_i}, where $i = 0, 1, \ldots, n$.

Commonalities Enhancement Module

To alleviate the problem of data scarcity in certain domains for a programming language, we design the pairwise commonality extraction module and the global commonality extraction module to capture the commonalities between language pairs and across all languages, respectively.

To extract commonalities between programming language i and other languages, we employ the pairwise commonality extraction module. This module utilizes the relevance between language i and other languages to identify shared features. Specifically, we calculate the relevance scores between the language i and representations of other languages as follows:

$$s_{ij} = (e_{L_i} W_i^T) \cdot (e_{L_j} W_j^T), \tag{4}$$

where the W_i and W_j are the parameters specific to programming language i and j, respectively, and $\cdot$ is the dot product operation. we utilize the relevance scores as weights to aggregate the commonalities from other languages for programming language i as follows:

$$\widetilde{e}_{L_i} = \sum_{j=1,2,\ldots,N, j\neq i} \left(\frac{e^{s_{ij}}}{\sum_{k=1,2,\ldots,N,k\neq i} e^{s_{ik}}} \cdot e_{L_j} \right). \tag{5}$$

To extract the overall commonality across all languages, we draw inspiration from the work presented in (Ou et al. 2021). In this approach, we aim to maximize the mutual information between the embedding of the $[CLS]$ token and the embedding of each language token. To estimate this mutual information, we utilize the Jensen-Shannon divergence estimator (JSDE) (Nowozin, Cseke, and Tomioka 2016), which is a widely used method and is insensitive to the number of negative samples. We apply JSDE to estimate mutual information and optimize it with model parameters. Specifically, we estimate the mutual information among all languages by using the following function:

$$\begin{aligned} \tilde{I}_\phi\left(e^c_{CLS}\right) = \sum_{i=1}^{N} (&E_{P(e^c_{CLS}, e_{L_i})}\left[D_\delta\left(e^c_{CLS}, e_{L_i}\right)\right] - \\ &E_{P(e^c_{CLS})P(e_{L_i})}\left[D_\delta\left(e^c_{CLS}, e_{L_i}\right)\right]), \end{aligned} \tag{6}$$

where $P(e^c_{CLS}, e_{L_i})$ represents the joint probability distribution, $P(e^c_{CLS})P(e_{L_i})$ represents the marginal probability, softplus function is defined as softplus(x) $= \log(1+e^x)$, and $D_\delta(\cdot,\cdot)$ is a discriminator realized by a neural network with parameter δ. By maximizing the mutual information using this approach, we aim to capture the commonalities that exist across all languages.

Finally, we employ a fusion strategy to combine the language token representation e_{L_i}, pairwise commonality representations $\widetilde{e}_{L_i}$, and the $[CLS]$ token representation e^c_{CLS} into a unified representation. This unified representation is then fed into a Multilayer Perceptron (MLP) to generate the final representation $\hat{e}_c$ of the code snippet c:

$$\hat{e}_c = \text{MLP}(e_{L_i} \circ \widetilde{e}_{L_i} \circ e^c_{CLS}). \tag{7}$$

By explicitly modeling the commonalities between language pairs and the overall commonalities across all languages, this method enhances the final representation of the code snippet in the data-scarce domain.

Confusion-Matrix-Guided Sampling Algorithm

To capture the specialties features of each programming language, we design a novel negative sampling algorithm based on contrastive learning. As illustrated in Figure 2(c), we use the confusion matrix between programming languages to determine the number of samples to be included for each type when constructing a batch. This enables us to group together languages that are often confused with each other in the same batch for contrastive learning, thereby automatically learning the characteristic features of different languages and better distinguishing the target language of the query.

Specifically, we initialize the confusion matrix $\mathbf{C} \in R^{n\times n}$ by summing an identity matrix $\mathbf{I}$ with an all-ones matrix $\mathbf{1}$ (i.e., $\mathbf{C} = \mathbf{I} + \mathbf{1}$) to achieve uniform sampling across other programming languages. During the subsequent training process, we employ a method similar to computing Mean Reciprocal Rank (MRR) to calculate the confusion matrix on the validation set. Specifically, at regular training intervals, for the calculation of C_{ij}, we begin by considering each query of the i-th language in the validation set Q^{val}_i. Using the current model, we retrieve the top K code snippets ($D_K = \{d_1, d_2, \ldots, d_K\}$) that are most similar to

the query. Subsequently, we calculate the reciprocal sum of the ranking values $rank_d$ for all code snippets d of the j-th language among the retrieved snippets. Then we update the confusion matrix $\mathbf{C}_{ij}$ as follows:

$$\mathbf{C}_{ij} = \sum_{q \in Q_i^{val}} \left(\sum_{d \in \{d | d \in D_K, L(d)=j\}} \frac{1}{rank_d} \right), \quad (8)$$

where $L(d)$ is the programming language of code snippet d.

Next, we use the idea of stratified sampling algorithm to guide the sampling process. Specifically, we determine the probability of selecting each programming language as $\mathbf{p}^l = \{p_1^l, p_2^l, ..., p_N^l\}$, where p_i^l is the probability to select the i-th programming language and is defined as:

$$p_i^l = \frac{N_i}{\sum_{j=0}^{N} N_j}, \quad (9)$$

Here, N_i is the number of the i-th programming language in the training set. Then, we calculate the confusion vector of language i using the confusion matrix:

$$\mathbf{v}_i = \mathbf{C}_{i\cdot}^{\top} + \mathbf{C}_{\cdot i}, \quad (10)$$

where $\mathbf{C}_{i\cdot}$ represents the scenario where queries with the target language being i are confused with each language which can be considered as precision and $\mathbf{C}_{\cdot i}$ represents the situation where queries with each language as the target language are confused with language i which can be considered as recall. By considering both the precision and recall of language i, we can comprehensively measure the confusion level between languages. Then, we use sun normalization $norm(\cdot)$ to normalize the $\mathbf{v}_i$ to obtain the sampling probability $\mathbf{p}_i = norm(\mathbf{v}_i)$, $\mathbf{p}_i = \{p_{i1}, p_{i2}, ..., p_{in}\}$ for language i, where p_{ij} is the probability of a query with the target language being the i-th one being confused with the j-th language. Besides, considering only the confusion level may result in fewer samples being sampled for some easily confused base languages. Thus we further use the base language probability boost to increase the probability of base language i being sampled as follows:

$$p'_{ii} = p_{ii} + (\alpha - p_{ii})^{\beta}, \quad (11)$$

where α is the threshold and β represents the final sampling probability. When the probability of base language i is lower than the threshold α, its probability is increased. Due to the sample probability p_{ii} is changed to p'_{ii}, we perform normalization on $\{p_{i1}, ..., p_{ii}, ...p_{in}\}$ to get the final probability $\{\hat{p}_{i1}, ..., \hat{p}'_{ii}, ...\hat{p}_{in}\}$ and construct next batch set $D_{sel} \subseteq (Q, C)$ according to each language's probability $\hat{p}_{ij}$.

Furthermore, the aforementioned sampling process may introduce language imbalance and intra-language imbalance. First, the inconsistent sampling quantity for each language compared to the actual language data distribution in the subsequent sampling process leads to language imbalance. To address this, we balance the probability of selecting the language $\mathbf{p}^l$ using the actual sampling probability:

$$\hat{\mathbf{p}}^l = norm(\mathbf{p}^l \cdot norm(\frac{\mathbf{p}^l}{\mathbf{p}^r})), \quad (12)$$

Language	Training	Validation	Test	Codebase
Ruby	2.5K	1.4K	1.2K	4.4K
JavaScipt	5.8K	3.9K	3.3K	13.9K
Go	16.7K	7.3K	8.1K	28.1K
Python	25.2K	13.9K	14.9K	43.8K
Java	16.4K	5.2K	10.9K	40.3K
PHP	24.1K	13.0K	14.0K	52.7K

Table 1: CodeSearchNet dataset statistics.

Here, $\mathbf{p}^r$ denotes the actual sampling probability for each language, which is determined during the sampling process and subsequently calculated. Besides, the repeated sampling of already sampled samples based on $\hat{\mathbf{p}}^l$ causes intra-language imbalance. To mitigate this, we apply exponential decay to the sampling probability of the already sampled samples to reduce the probability of resampling them.

Finally, we conduct contrastive learning based on the constructed batch D_{sel}:

$$\mathcal{L} = \sum_{(q,c) \in (Q,C)} \log \frac{exp(\varphi(e^q_{CLS}, \hat{e}_c))}{\sum_{(q',c') \in D_{sel}} exp(\varphi(e^{q'}_{CLS}, \hat{e}_{c'}))}, \quad (13)$$

where $\varphi(\cdot, \cdot)$ is used to measure the cosine similarity between the query and code representations. For negative samples of the same language, the primary objective of contrastive learning is to amplify the semantic gap between the query and code. For negative samples of different languages, contrastive learning not only increases the semantic gap between the query and code in terms of meaning but also accentuates the differences in language-specific features. As a result, contrastive learning facilitates more effective modeling of linguistic characteristics.

The advantage of this algorithm is that it can construct negative samples according to the confusion matrix, thus automatically learning the differences between easily confused languages. Furthermore, it can automatically construct the confusion matrix during the evaluation period, without consuming additional computational costs for model inference.

Experiment

In this section, we conduct experiments on monolingual and multilingual tasks with a real-world code retrieval dataset, to verify the effectiveness of our proposed approach.

Experimental Setup

Dataset. Since we need to evaluate model performance in a multilingual environment, we have chosen CodeSearchNet (Husain et al. 2019) as our dataset. This dataset collects code snippets and queries related to six programming languages (Go, Python, Java, JavaScript, Ruby, and PHP) from GitHub, and it is the largest and most widely used dataset for assessing code retrieval performance. Table 1 contains the statistics for the dataset.

Evaluation Tasks. We conduct model performance evaluation in both monolingual and multilingual scenarios. In the monolingual scenario, we evaluate the model on test sets for

Framework	Model	Ruby	JavaScript	Go	Python	Java	PHP	Overall
Multilingual	RoBERTa(code)	46.0	46.3	82.1	54.7	56.1	52.3	56.2
	CodeBERT	50.2	50.1	83.8	59.2	59.9	55.6	59.8
	GraphCodeBERT	51.7	51.4	84.6	62.6	61.4	58.6	61.7
	UniXCoder	57.1	56.8	86.4	65.3	65.6	60.3	65.3
Distill	RoBERTa(code)	45.3	46.5	81.6	56.0	57.7	53.5	56.8
	CodeBERT	49.8	49.1	82.2	60.9	61.8	57.1	60.2
	GraphCodeBERT	51.3	50.3	84.0	64.2	63.6	60.3	62.3
	UniXCoder	58.2	58.9	88.2	67.0	67.3	61.9	65.4
CONSIDER	RoBERTa(code)	**52.6(+6.6)**	**53.4(+7.1)**	**87.4(+5.3)**	**60.2(+5.5)**	**62.7(+6.6)**	**59.1(+6.8)**	**62.6(+5.8)**
	CodeBERT	**56.2(+6.0)**	**57.0(+6.9)**	**89.0(+5.2)**	**64.6(+5.4)**	**66.5(+6.6)**	**62.3(+6.7)**	**65.9(+6.1)**
	GraphCodeBERT	**57.8(+6.1)**	**57.6(+6.2)**	**89.5(+4.9)**	**66.1(+3.5)**	**67.3(+5.9)**	**63.1(+4.5)**	**66.9(+5.2)**
	UniXCoder	**61.6(+3.4)**	**60.9(+2.0)**	**90.2(+2.0)**	**69.7(+4.4)**	**69.9(+4.3)**	**65.0(+4.7)**	**69.6(+4.3)**

Table 2: Comparison of the overall performance between our framework and the baseline in a multilingual scenario, measured by the MRR metric. Bold means state of the art on this metric.

each individual language. To simulate the multilingual context of the real world, we have merged the test sets of each language to serve as the test set in a multilingual scenario. We use mean reciprocal rank(MRR) (Hull 1999) as evaluation metrics for all models.

$$MRR = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{rank_i}. \tag{14}$$

Comparison Methods. We compared our framework with other multilingual retrieval model training methods: 1). **Multilingual**: Mixing training datasets from various programming languages, followed by applying contrastive learning algorithms for training. 2). **Distill**: Utilizing the multilingual retrieval model training framework proposed by (Li, Xu, and Chen 2022): first, training monolingual teacher models for each programming language, and then training multilingual student models in a multilingual environment. Moreover, to demonstrate that our multilingual retrieval framework is model-agnostic and can be applied to different pre-trained Transformers to achieve better performance, we fine-tuned four popular pre-trained code retrieval Transformers—RoBERTa(code) (Husain et al. 2019), CodeBERT (Feng et al. 2020), GraphCodeBERT (Guo et al. 2021), and UniXCoder (Guo et al. 2022)—on the CodeSearchNet dataset in our study.

Implementation Details

Our CONSIDER framework is implemented in PyTorch. For all models, we map the final output dimensions to 768, utilizing the AdamW optimizer (Loshchilov and Hutter 2017). Batch size, learning rate, and training steps are set to 256, 2e-5, and 50K respectively. The maximum sequence lengths for text and code are set to 128 and 320 respectively. All experiments are conducted using two Tesla A100 GPUs. We consider hyperparameters α within $\{0.5, 0.6, 0.7\}$ and β within $\{1.5, 1.75, 2.0\}$. We conduct a grid search across various scenarios to identify their optimal combinations.

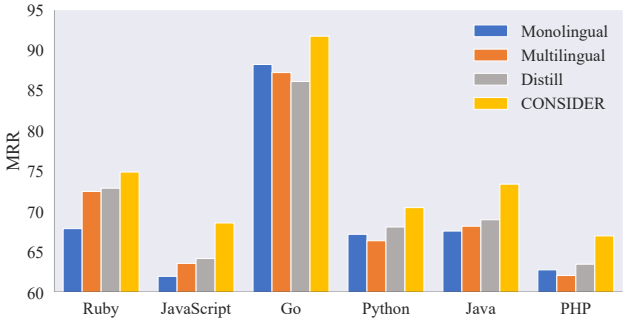
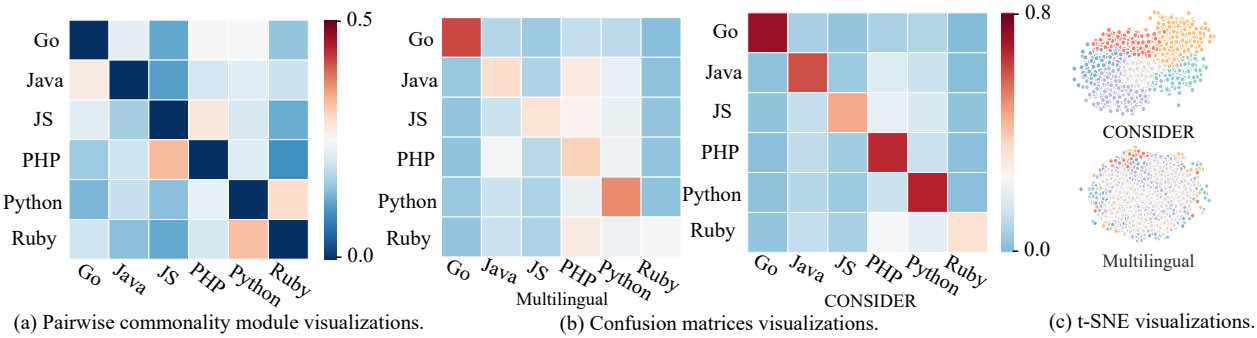


Figure 3: Comparison of the overall performance between our framework and the baseline in a monolingual scenario.

Overall Results

First, we test all multilingual frameworks in a multilingual scenario, as shown in Table 2. We find that, compared to the monolingual scenario, both directly applying a multilingual training set and adopting the knowledge distillation learning method result in a more significant performance decline. However, the performance degradation of our proposed CONSIDER framework in a multilingual scenario is relatively smaller. This indicates that our method better identifies users' language intent in multilingual scenarios by modeling language specialties.

Next, we conduct experiments in a monolingual scenario, as shown in Figure 3. We find that although we directly apply a multilingual training set to improve the overall performance of the model, especially for low-resource languages, it affects the performance of some high-resource languages. Although the knowledge distillation learning method enhances the overall performance, the improvement is relatively small due to the limitations of the teacher model. In contrast, our proposed CONSIDER framework demonstrates a stable improvement in performance across all languages compared to the monolingual model, suggesting that our CONSIDER framework leverages the similarities between programming languages to enhance the performance of multiple languages.



(a) Pairwise commonality module visualizations. (b) Confusion matrices visualizations. (c) t-SNE visualizations.

Figure 4: Visualization of commonalities and specialties modeling effectiveness in CONSIDER.

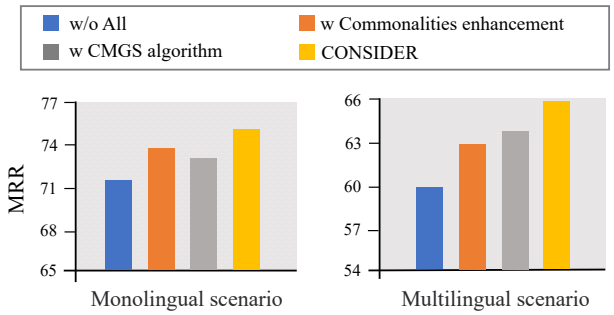


Figure 5: Ablation experiments.

Model Analysis

Ablation Study. To investigate the impact of each module, we conducted an ablation study. We use the CodeBERT model to initialize our framework and conduct experiments in monolingual versus multilingual scenarios. The results are shown in Figure 5. We observed that removing a module leads to a decrease in model performance, indicating the effectiveness of our design. Furthermore, we noticed that the commonalities enhancement module enhances the overall performance. Modeling language-specific features produces more significant improvements in performance in multilingual scenarios. Moreover, we found that employing the CMGS algorithm also improves the model performance in monolingual scenarios. We speculate that this is because the update of the confusion matrix has led to a more diversified sample distribution in contrastive learning.

Visualization. Firstly, to examine the effectiveness of our framework in modeling language features, we statistically analyze and visualize the correlation scores for each pair of languages in the pairwise commonality extraction module, as depicted in Figure 4(a). We observe that the language pairs Javascript and PHP, and Python and Ruby have strong correlations (Javascript and PHP are commonly used in web development). This result confirms the commonalities enhancement module can exploit the commonalities between languages to improve their performance.

Next, to investigate the effectiveness of our framework in modeling language features, we apply the Multilingual and CONSIDER frameworks to perform multilingual fine-tuning of the CodeBERT model. We visualize the confusion matrix of the model on the test set. As shown in Figure 4(b), we notice that applying this algorithm leads to a confusion matrix closer to a diagonal matrix. Additionally, to visually demonstrate CONSIDER's ability to model language features, we randomly select 500 code snippets from the test set of each language and project their representations onto a 2D space using t-SNE (Van der Maaten and Hinton 2008). In Figure 4(c), we differentiate code representations of different programming languages using distinct colors, and we compare our framework with the multilingual method. Upon observation, we find that the visualization plot using the CONSIDER framework shows clearly separated clusters of different languages, while the plots using other frameworks depict larger areas of overlap among code representations of various programming languages. This indicates that CONSIDER can effectively model the characteristics of programming languages, whereas other multilingual retrieval frameworks struggle to accurately capture the specialties features of different programming languages.

Conclusion

In this paper, we investigated the task of multilingual code retrieval. We proposed a novel multilingual code retrieval framework CONSIDER to enhance the capability of the retrieval model in multilingual scenarios. Specifically, we first modeled the commonalities between programming languages and enhanced the representation of each language based on this. Then, we introduced a novel confusion-matrix-guided sampling algorithm to model the specialties of languages. Through extensive experiments on both monolingual and multilingual retrieval scenarios, we demonstrated that CONSIDER could leverage the commonalities between programming languages to boost overall performance, and it could effectively model the specialties of languages, thereby enabling a better understanding of the target language of user queries. We also conducted additional analysis experiments to substantiate the effectiveness and rationality of CONSIDER.

Acknowledgements

This research was partially supported by grants from the National Key Research and Development Program of China (No. 2021YFF0901003), and the National Natural Science Foundation of China (No.62106244), the University Synergy Innovation Program of Anhui Province (No. GXXT-2022-042)

References

Arakelyan, S.; Hakhverdyan, A.; Allamanis, M.; Garcia, L.; Hauser, C.; and Ren, X. 2022. NS3: Neuro-symbolic Semantic Code Search. In *NeurIPS*.

Bui, N. D. Q.; Yu, Y.; and Jiang, L. 2021. Self-Supervised Contrastive Learning for Code Retrieval and Summarization via Semantic-Preserving Transformations. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, 511–521.

Cambronero, J.; Li, H.; Kim, S.; Sen, K.; and Chandra, S. 2019. When deep learning met code search. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, 964–974.

Chen, Q.; and Zhou, M. 2018. A neural framework for retrieval and summarization of source code. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, 826–831.

Clark, K.; Khandelwal, U.; Levy, O.; and Manning, C. D. 2019. What Does BERT Look At? An Analysis of BERT's Attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*.

Feng, Z.; Guo, D.; Tang, D.; Duan, N.; Feng, X.; Gong, M.; Shou, L.; Qin, B.; Liu, T.; Jiang, D.; and Zhou, M. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, 1536–1547.

Giorgi, J.; Nitski, O.; Wang, B.; and Bader, G. 2021. DeCLUTR: Deep Contrastive Learning for Unsupervised Textual Representations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

Guo, D.; Lu, S.; Duan, N.; Wang, Y.; Zhou, M.; and Yin, J. 2022. UniXcoder: Unified Cross-Modal Pre-training for Code Representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, 7212–7225.

Guo, D.; Ren, S.; Lu, S.; Feng, Z.; Tang, D.; Liu, S.; Zhou, L.; Duan, N.; Svyatkovskiy, A.; Fu, S.; Tufano, M.; Deng, S. K.; Clement, C. B.; Drain, D.; Sundaresan, N.; Yin, J.; Jiang, D.; and Zhou, M. 2021. GraphCodeBERT: Pre-training Code Representations with Data Flow. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*.

Haldar, R.; Wu, L.; Xiong, J.; and Hockenmaier, J. 2020. 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*, 8563–8568.

He, K.; Fan, H.; Wu, Y.; Xie, S.; and Girshick, R. 2020. Momentum Contrast for Unsupervised Visual Representation Learning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

He, L.; Huang, Z.; Chen, E.; Liu, Q.; Tong, S.; Wang, H.; Lian, D.; and Wang, S. 2023. An Efficient and Robust Semantic Hashing Framework for Similar Text Search. *ACM Trans. Inf. Syst.*, 41(4).

Huang, J.; Tang, D.; Shou, L.; Gong, M.; Xu, K.; Jiang, D.; Zhou, M.; and Duan, N. 2021. CoSQA: 20, 000+ Web Queries for Code Search and Question Answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, 5690–5700.

Hull, D. 1999. Xerox TREC-8 Question Answering Track Report.

Husain, H.; Wu, H.; Gazit, T.; Allamanis, M.; and Brockschmidt, M. 2019. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. *CoRR*, abs/1909.09436.

Kim, J.; Lee, S.; Hwang, S.; and Kim, S. 2010. Towards an Intelligent Code Search Engine. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*.

Kovaleva, O.; Romanov, A.; Rogers, A.; and Rumshisky, A. 2019. Revealing the Dark Secrets of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Lemos, O. A. L.; de Paula, A. C.; Zanichelli, F. C.; and Lopes, C. V. 2014. Thesaurus-based automatic query expansion for interface-driven code search. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, 212–221.

Li, H.; Miao, C.; Leung, C.; Huang, Y.; Huang, Y.; Zhang, H.; and Wang, Y. 2022. 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*, 4924–4936.

Li, W.; Xu, J.; and Chen, Q. 2022. Knowledge Distillation-Based Multilingual Code Retrieval. *Algorithms*, 15(1): 25.

Ling, X.; Wu, L.; Wang, S.; Pan, G.; Ma, T.; Xu, F.; Liu, A. X.; Wu, C.; and Ji, S. 2021. Deep Graph Matching and

Searching for Semantic Code Retrieval. *ACM Trans. Knowl. Discov. Data*, 15(5): 88:1–88:21.

Loshchilov, I.; and Hutter, F. 2017. Fixing Weight Decay Regularization in Adam. *CoRR*, abs/1711.05101.

Lv, F.; Zhang, H.; Lou, J.-g.; Wang, S.; Zhang, D.; and Zhao, J. 2015. CodeHow: Effective Code Search Based on API Understanding and Extended Boolean Model (E). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*.

Ma, Y.; Yu, Y.; Li, S.; Jia, Z.; Ma, J.; Xu, R.; Dong, W.; and Liao, X. 2023. MulCS: Towards a Unified Deep Representation for Multilingual Code Search. In *IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2023, Taipa, Macao, March 21-24, 2023*, 120–131.

Mukherjee, R.; Jermaine, C.; and Chaudhuri, S. 2020. Searching a Database of Source Codes Using Contextualized Code Search. *Proc. VLDB Endow.*, 13(10): 1765–1778.

Nie, L.; Jiang, H.; Ren, Z.; Sun, Z.; and Li, X. 2016. Query Expansion Based on Crowd Knowledge for Code Search. *IEEE Trans. Serv. Comput.*, 9(5): 771–783.

Ning, Y.; Huang, Z.; Lin, X.; Chen, E.; Tong, S.; Gong, Z.; and Wang, S. 2023. Towards a Holistic Understanding of Mathematical Questions with Contrastive Pre-training. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, 13409–13418.

Nowozin, S.; Cseke, B.; and Tomioka, R. 2016. f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 271–279.

Ou, Z.; Su, Q.; Yu, J.; Zhao, R.; Zheng, Y.; and Liu, B. 2021. Refining BERT Embeddings for Document Hashing via Mutual Information Maximization. In *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*, 2360–2369.

Shuai, J.; Xu, L.; Liu, C.; Yan, M.; Xia, X.; and Lei, Y. 2020. 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*, 196–207.

Tong, W.; Tong, S.; Hunag, W.; He, L.; Ma, J.; Liu, Q.; and Chen, E. 2020. Exploiting knowledge hierarchy for finding similar exercises in online education systems. In *2020 IEEE International Conference on Data Mining (ICDM)*, 1298–1303. IEEE.

Wan, Y.; Shu, J.; Sui, Y.; Xu, G.; Zhao, Z.; Wu, J.; and Yu, P. S. 2019. Multi-modal Attention Network Learning for Semantic Source Code Retrieval. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*, 13–25.

Yao, Z.; Peddamail, J. R.; and Sun, H. 2019. 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*, 2203–2214.

Ye, W.; Xie, R.; Zhang, J.; Hu, T.; Wang, X.; and Zhang, S. 2020. Leveraging Code Generation to Improve Code Retrieval and Summarization via Dual Learning. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, 2309–2319.

Zhang, F.; Niu, H.; Keivanloo, I.; and Zou, Y. 2018. Expanding Queries for Code Search Using Semantically Related API Class-names. *IEEE Transactions on Software Engineering*, 44(11): 1070–1082.

Zhao, C.; Zhao, H.; He, M.; Zhang, J.; and Fan, J. 2023. Cross-domain recommendation via user interest alignment. In *Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*, 887–896.

Zubkov, M.; Spirin, E.; Bogomolov, E.; and Bryksin, T. 2022. Evaluation of Contrastive Learning with Various Code Representations for Code Clone Detection. *CoRR*, abs/2206.08726.