MD3R: Minimizing Data Distribution Discrepancies to Tackle Inconsistencies in Multilingual Query-Code Retrieval

Aofan Liu[1,2], Yuguo Yin[2], Hongjian Xing[2], Zhen Li[3], and Yiyan Qi ✉[1]

[1]International Digital Economy Academy (IDEA)
[2]School of Electronic and Computer Engineering, Peking University
[3]The Hong Kong Polytechnic University

Abstract

Multilingual Code Retrieval (MLCR) is a critical task for supporting modern software development workflows that increasingly involve multiple programming languages. While existing methods have shown progress, MLCR still faces two core challenges: firstly, the data distribution discrepancy caused by training on single query-monolingual code pairs leads to inconsistency in cross-lingual retrieval; secondly, the data scarcity of certain languages in specific domains limits the effectiveness of consistent representation learning. To address these issues, we first analyze the inconsistency from two perspectives: modality alignment direction error and model weight error. We derive an upper bound for the weight error to quantify the impact of inconsistency and find that this upper bound primarily stems from data distribution discrepancies during the training process. Based on this theoretical analysis, we propose a novel **Cross-lingual Consistent MLCR scheme** call MD3R (Minimizing Data Distribution Discrepancies in Retrieval). Our scheme employs tailored contrastive learning strategies, including **co-anchor contrastive learning (CACL)** and **1-to-k contrastive learning (KCL)**, aimed at mitigating the impact of data distribution bias, thereby enhancing cross-lingual embedding alignment and retrieval consistency. In the widely used CodeSearchNet benchmark, our method achieves breakthroughs in both retrieval recall and consistency metrics across six mainstream programming languages, including python, attaining state-of-the-art performance.

1 Introduction

The continuous advancement of programming technologies and the flourishing development of the open-source community have driven significant changes in software development. Previously, the environment followed an earlier paradigm based on single programming languages. Now, it is a complex paradigm that involves collaborative work across multiple programming languages (Li et al., 2024). An increasing number of software projects integrate disparate programming languages, presenting developers with challenges in searching, understanding, reusing, and modifying code across language boundaries (Allal et al., 2023).

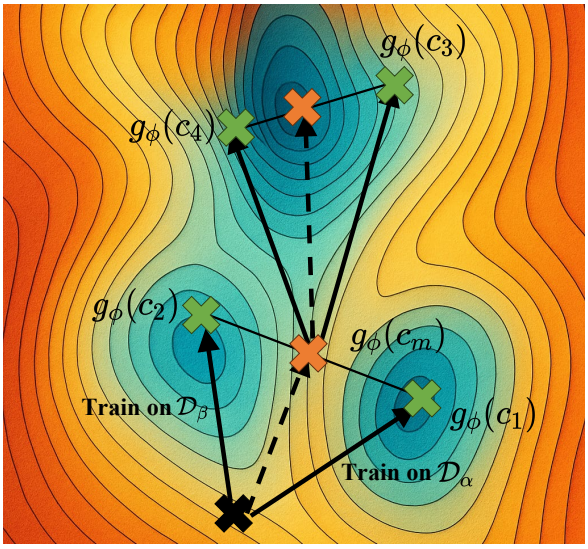

Figure 1: The training performance on the dataset $\mathcal{D}_\alpha$, corresponding to the vector direction $\mathbf{g}_\phi(\mathbf{c_1})$, is worse than on the intermediate dataset $\mathcal{D}_\gamma$, where $\mathbf{g}_\phi(\mathbf{c_m}) = \frac{\mathbf{g}_\phi(\mathbf{c_1})+\mathbf{g}_\phi(\mathbf{c_2})}{2}$. Here, $\mathbf{g}_\phi(\mathbf{c_1})$ and $\mathbf{g}_\phi(\mathbf{c_2})$ are the vector forms of embeddings, and $\mathcal{D}_\alpha$ and $\mathcal{D}_\beta$ correspond to the respective datasets.

For instance, within a typical project, backend logic may be implemented in Java, the frontend interactive interface constructed using Vue, while foundational service modules and even machine learning components might be authored in Python or C++. In such multilingual, heterogeneous environments, developers frequently need to retrieve functionally analogous code snippets across linguistic divides to enhance development efficiency and mitigate code redundancy.

As shown in Fig. 1, although existing MLCR schemes are inherently multilingual, the alignment process becomes more complex when handling

multiple languages simultaneously. The ideal alignment process in the MLCR task should ensure precise semantic correspondence between the query and the multilingual programming code tokens, thereby facilitating the selection of the best match. However, current schemes arbitrarily pair query with instances of only one programming language during each epoch, which impairs the establishment of effective co-alignment across all languages. This leads to inconsistencies in the modality matching of multilingual code and query in existing schemes: query-code instances exhibit inconsistent similarity rankings across languages, and it is difficult to obtain the same retrieval results in different languages.

To address the pervasive inconsistency issues in MLCR systems, we conduct a detailed analysis of inconsistency from two perspectives: modal alignment directional error and weight error. We further derived an theoretical upper bound for weight error to quantify the degree of inconsistency during model inference. Through theoretical analysis of the upper bound on weight error, we identify that the inconsistency primarily stems from data distribution Discrepancies introduced by random sampling in the training processes of existing schemes.

In response to this identified root cause, we introduce a novel scheme, MD3R, specifically engineered to mitigate the inconsistency arising from data distribution issues. Within the MD3R scheme, we explore two distinct training strategies tailored for different application scenarios: one prioritizing high performance and the other prioritizing low computational overhead. For scenarios demanding high performance, we design 1-to-K Contrastive Learning (KCL). KCL functions by substantially mitigating intra-epoch data distribution error, thereby reducing the disparity between the empirically learned model parameters and the theoretically optimal weights. This mechanism facilitates enhanced multilingual query-code alignment fidelity, leading to a marked improvement in retrieval performance, specifically in terms of recall and consistency. Conversely, for scenarios prioritizing low computational overhead, query-code Co-Anchor Contrastive Learning (CACL) is utilized. CACL refines the alignment process by establishing code and query as co-anchors for other languages, rectifying potential deviations in alignment direction and further ameliorating overall data distribution discrepancies. Crucially, CACL achieves these benefits, contributing to improved consistency and recall, while imposing only approximately 10% additional computational overhead (memory and time).

Our Contribution are as follows:

1. We derive a theoretical upper bound for weight error. Our analysis further reveals that discrepancies in training data distribution serve as the key factor contributing to the inconsistencies observed in existing ML-CLAP methods.

2. To address the inconsistency problem, we proposes the MD3R framework, which minimize data distribution discrepancies to improve the consistency of multilingual query-code retrieval performance. The MD3R framework contains two training strategies: KCL (suit for retrieval performance-first scenarios) and CACL (suit for training overhead-first scenarios).

3. Experiments conducted on the ChatGPT-translated CodeSearchNet datasets demonstrate that both CACL and KCL can effectively enhance the performance and consistency of multilingual query-code retrieval. Notably, KCL achieved leading results in both monolingual MLCR and multilingual MLCR tasks.

2 Related Work

Research concerning code retrieval falls into two principal areas: monolingual and multilingual model development. Recently, how to more accurately understand the developer's search intent within a specific programming context has also become a new research focus.

Monolingual Code Retrieval Methods Research on monolingual code retrieval has explored diverse approaches to identifying relevant code within a single programming language. One stream of research has focused on query enhancement, aiming to enrich user queries with additional semantic information to improve matching accuracy (Lemos et al., 2014; Lv et al., 2015; Zhang et al., 2017; Arakelyan et al., 2022). Another significant direction involves multi-perspective modeling of the code, which leverages structural code features from Abstract Syntax Trees (ASTs), Control Flow Graphs (CFGs), and Data Flow Graphs (DFGs) to better capture code semantics (Kim et al., 2010;

Chen and Zhou, 2018; Zubkov et al., 2022). Furthermore, multitask learning has been employed to enhance retrieval capabilities by designing auxiliary tasks, such as annotation generation and code generation (Yao et al., 2019; Ye et al., 2020; Feng et al., 2020). More recently, addressing limitations of prior methods, Dong et al. (Dong et al., 2025) emphasized programming context (e.g., existing code) for understanding search intent. They introduced the CodeSearchNet-C dataset and ConCR framework, using Context Walking (CW) and a Context Hierarchical Encoder (CHE) to model context and developer habits, thus improving intent understanding.

Multilingual Code Retrieval Methods The growing prevalence of multilingual software projects has sparked an increasing demand for code retrieval systems that can operate across different programming languages. One method employs knowledge distillation, 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 (Li et al., 2022). Another approach involves using compilers like LLVM to pre-generate a consistent intermediate representation (IR) for each programming language, which can obtain a unified representation across languages (Lattner and Adve, 2004; Puri et al., 2021). However, these methods often overlook modeling the specialties among programming languages, leading to suboptimal performance in scenarios with sparse programming language data, and a struggle to identify language intent from linguistic features within user queries. To address these issues, Li et al. (Li et al., 2024) proposed the CONSIDER framework, which models pairwise and global commonalities to enhance representation and uses a novel Confusion-Matrix-Guided Sampling Algorithm in contrastive learning to capture language specialties, thereby improving the discernment of query intent.

3 Problem Formulation and Definition

MLCR is a significant and challenging cross-modal retrieval task that aims to bridge the gap between natural language and code in different programming languages. This requires understanding the intent of a natural language query and aligning it with the semantics of code in various programming languages.

3.1 Formal Description of MLCR

MLCR aims to retrieve semantically relevant code snippets from a dataset containing code in multiple programming languages, based on a given natural language query. We formally define the multilingual code search dataset as $D = \{(q_i, c_{i1}, \ldots, c_{iK})\}_{i=1}^{N}$, where N is the size of the dataset, K is the total number of programming languages, q_i is the natural language query corresponding to the i-th data instance, and c_{ik} is the code snippet in the k-th programming language associated with query q_i in the i-th data instance.

We typically use a natural language query encoder $f_\theta(\cdot)$ and a multilingual code encoder $g_\phi(\cdot)$ to embed queries and code into a shared joint vector space. Ideally, for related query q_i and code c_{ik}, their distance in the embedding space should be small (or similarity should be high).

Borrowing the idea of contrastive learning, the ideal query-code alignment probability distribution can be expressed as:

$$p(q_i, c_{ik}) = \frac{\exp(s(f_\theta(q_i), g_\phi(c_{ik}))/\tau)}{\sum_{j=1}^{N}\sum_{l=1}^{K}\exp(s(f_\theta(q_j), g_\phi(c_{jl}))/\tau)}, \quad (1)$$

where $s(\cdot)$ denotes the similarity function between query and code embeddings (e.g., cosine similarity), and τ is the temperature parameter.

The optimization objective for learning the ideal embedding space is to minimize the negative log-likelihood:

$$\min_{\theta,\phi} \sum_{i=1}^{N}\sum_{k=1}^{K} p(q_i, c_{ik}) E_{(q_i, c_{ik})}[-\log p(q_i, c_{ik})]. \quad (2)$$

However, in the training of practical MLCR models, due to limitations in computational resources or training strategies, it may not be feasible to process code from all languages simultaneously within each training batch (Monteiro et al., 2023). A common simplification is to randomly select code from one programming language for each query in each batch for training. For each epoch e, a set of language is randomly selected: $Q = \{q_1, \ldots, q_N\}$, where $q_i \leftarrow \{1, \ldots, K\}$. At this point, the probability distribution used in training becomes:

$$p'_e(q_i, c_{iq_i}) = \frac{\exp(s(f_\theta(q_i), g_\phi(c_{iq_i}))/\tau)}{\sum_{j=1}^{N}\exp(s(f_\theta(q_j), g_\phi(c_{jq_j}))/\tau)}, \quad (3)$$

and the corresponding optimization function is:

$$\min_{\theta,\phi} \sum_{i=1}^{N} p'_e(q_i, c_{iq_i}) E_{(q_i, c_{iq_i})}[-\log p'_e(q_i, c_{iq_i})]. \quad (4)$$

Since the distribution used in training $p'_e(q_i, c_{iq_i})$ is not the same as the ideal distribution $p(q_i, c_{ik})$, this can lead to the learned model failing to fully capture the alignment between natural language queries and code in all programming languages, thereby introducing problems.

3.2 Analysis of the Inconsistency Issue

We found that training models with existing approaches can lead to cross-language inconsistency issues. Specifically, for the same natural language query, the retrieval rank of relevant code snippets in different programming languages may show significant variations.

To address the inconsistency issue, we analyze its causes from the perspectives of modality alignment direction error and weight error. We further derive the upper bound of error between the model weights and the optimal weights, the analytical findings are as follows:

3.2.1 Modality Alignment Direction Error

[**Optimal direction of modality alignment**] Given the query embedding $f_\theta(q)$ and the multilingual embedding of K code $\{g_\phi(c_1), ..., g_\phi(c_K)\}$, the optimal alignment direction for the query embedding $f_\theta(q)$ is the arithmetic mean of all language text embeddings.

Proof. The total distance between the code embedding and each language embedding needs to be minimized so that the embedding space for different languages is similar to the code embedding and consistent cross-lingual retrieval results are achieved. The loss function to obtain the minimization of the total distance is shown below:

$$\mathcal{L} = \sum_{k=1}^{K} ||f_\theta(q) - g_\phi(c_k)||^2. \tag{5}$$

Take the partial derivative of the loss function L and take 0,

$$\begin{aligned} \frac{\partial L}{\partial f_\theta(q)} &= \sum_{k=1}^{K} 2(f_\theta(q) - g_\phi(c_k)) = 0, \\ f_\theta(q) &= \frac{1}{K}\sum_{k=1}^{K} g_\phi(c_k). \end{aligned} \tag{6}$$

It's obvious that the loss function $\mathcal{L}$ takes the minimum value when the alignment direction of the query is the arithmetic mean of $\{c_1, \ldots, c_K\}$, allowing the model to achieve the most consistent retrieval in theory. □

The existing scheme randomly selects a code snippet and query, and this random selection leads to the alignment direction deviating from the optimal one, which results in differences in the degree of alignment between different programming languages and query, and ultimately triggers the inconsistency problem.

3.2.2 Weight Error

To investigate the upper limit of weight deviation, we consider stochastic gradient descent (SGD) as the foundational optimization framework for heuristic examination and derives a weight error upper bound.

[**Weight error upper bound**] Let T represent the fixed number of training iterations per epoch. The modified data distribution generated through randomized language sampling under the existing MLCR scheme is designated as p'_i in epoch i, while p signifies the native data distribution. We define $\mathbf{w}'_{eT}$ as the parameter vector obtained at the T-th iteration during the e-th epoch under p'_e, contrasting with $\mathbf{w}'_{eT}$ derived from training with the original distribution p. Under the premise that the gradient operator $\nabla_{\mathbf{w}} E_{(q,c)}[log\ p(q,c)]$ satisfies $\lambda_{(q,c)}$-Lipschitz (Bethune et al., 2024), the subsequent inequality establishes the upper bound for weight error:

$$\begin{aligned} ||\mathbf{w}_{eT} - \mathbf{w}'_{eT}|| \leq \eta \sum_{i}^{e} \sum_{(q,c)} ||p'_i(q,c) - p(q,c)|| \\ \times \Big(\sum_{j=(i-1)T}^{iT-1} a_i^j g_{max}(\mathbf{w}_{iT-1-j}) \Big) \end{aligned} \tag{7}$$

$$g_{max}(\mathbf{w}) = max_{(q,c)} ||\nabla_{\mathbf{w}} E_{(q,c)}[log\ p(q,c)]||, \tag{8}$$

$$a_i = 1 + \eta \sum_{(q,c)} p'_i(q,c)\lambda_{(q,c)}. \tag{9}$$

The model weight $\mathbf{w}$ comprises two components: query encoder weight θ and multilingual code encoder weight ϕ. We adopt (q, c) as a shorthand notation for all query-code pairs within a training batch at step T, where code t may appear in arbitrary languages. The term $\sum_{(q,c)} ||p(q,c) - p'_i(q,c)||$ denotes batch-level distribution discrepancy at training epoch i.

Conceptually, the weight error originates from data distribution divergence discrepancies $\sum_{(q,c)} ||p(q,c) - p'_i(q,c)||$ in each epoch i. Given

the amplification factor $a \geq 1$, both error components demonstrate epoch-/step-dependent accumulation. Notably, this discrepancy magnitude further correlates with hyperparameters including the learning rate η, the total training steps T, and the supremum gradient $g_{max}(\mathbf{w}_{eT-1-j})$. The detailed proof of formula (7) is shown in Appendix.

4 Proposed Multilingual Code Retrieval Scheme

Addressing the inconsistencies observed in existing MLCR methods when matching similar instances across different languages, which we attribute to data distribution biases caused by random language sampling during training, we propose a unified MLCR framework. This framework includes two training strategies designed to mitigate these biases and enhance embedding alignment and retrieval consistency. Fig. 2 clearly highlights the key differences by contrasting traditional alignment with our proposed KCL and CACL methods.

4.1 1-to-K Contrastive Learning (KCL)

The basic idea behind using 1-to-K contrastive learning is that instead of randomly sampling languages for each code snippet, the original dataset is directly used to train the MLCR model. Theoretically, such a method will make the data error equal to 0, better align the embedding space of code and multilingual text, and improve consistency and recall. The loss function $\mathcal{L}_{kcl}^{ct}$ of proposed 1-to-K contrastive learning method in MLCR is shown below:

$$\mathcal{L}_{kel} = \frac{1}{2NK}(\mathcal{L}_{kcl}^{c2q} + \mathcal{L}_{kcl}^{q2c}) \tag{10}$$

where $\mathcal{L}_{kcl}^{c2q}$ denotes the contrastive learning loss function from multilingual code to query, and $\mathcal{L}_{kcl}^{q2c}$ denotes the contrastive learning loss function from query to multilingual code. K is the number of languages and N is the number of data instances. In practical MLCR applications, supporting more languages amplifies these overheads compared to existing schemes (Wang et al., 2023).

$$\mathcal{L}_{kcl}^{c2q} = -\sum_{k=1}^{K}\sum_{i=1}^{N}\log\frac{\exp(s(f_\theta(q_i), g_\phi(c_{ik}))/\tau)}{\sum_{j=1}^{N}\exp(s(f_\theta(q_i), g_\phi(c_{jk}))/\tau)}, \tag{11}$$

where $\mathcal{L}_{kcl}^{c2q}$ denotes the contrastive learning loss function from multilingual code to query.

$$\mathcal{L}_{kcl}^{q2c} = -\sum_{k=1}^{K}\sum_{i=1}^{N}\log \frac{\exp(s(g_\phi(c_{ik}), f_\theta(q_i))/\tau)}{\sum_{j=1}^{N}\exp(s(g_\phi(c_{ik}), f_\theta(q_j))/\tau)} \tag{12}$$

where $\mathcal{L}_{kcl}^{q2c}$ denotes the contrastive learning loss function from multilingual query to code. K is the number of languages and N is the number of data instances. In practical MLCR applications, supporting more languages amplifies these overheads compared to existing schemes. To address this, we further propose CACL, which improves retrieval consistency and recall without significantly increasing overhead.

However, KCL requires simultaneous processing of code and query in multiple languages during training, which substantially increases GPU memory usage and training time overhead, particularly in scenarios supporting a larger number of code languages.

4.2 Query-Code Co-Anchor Contrastive Learning (CACL)

To reduce the weighting error with as little increase in training time and GPU memory consumption as possible, we propose query-code co-anchor contrastive learning (CACL). During the training process, each data takes its code snippet, text, and query in other random languages and does contrastive learning with each other. For each epoch, given a set of random numbers $Q = \{q_1, ...q_N\}$, $q_i \leftarrow \{1, ...K\}$, get the triplet of the training data (c_i, q_{i1}, q_{iq_i}), where c_i denotes i-th code snippet, q_{i1} denotes the query in a primary language (e.g., Python), and q_i denotes the query of i-th language. We have the training loss $\mathcal{L}_{cacl}$ shown below:

$$\mathcal{L}_{cacl} = \frac{1}{6N}(\mathcal{L}_{cacl}^{cq_1} + \mathcal{L}_{cacl}^{cq_{rand}} + \mathcal{L}_{cacl}^{q_1 q_{rand}}). \tag{13}$$

The loss function $\mathcal{L}_{cacl}$ consists of three components $\mathcal{L}_{cacl}^{cq_1}$, $\mathcal{L}_{cacl}^{cq_{rand}}$, $\mathcal{L}_{cacl}^{q_1 q_{rand}}$. All three components are based on the following general contrastive learning loss formulation:

$$\begin{aligned}\mathcal{L}_{cacl}^{cq} = &-\sum_{i=1}^{N}\log\frac{\exp(s(u_i, v_i)/\tau)}{\sum_{j=1}^{N}\exp(s(u_i, v_j)/\tau)} \\ &-\sum_{i=1}^{N}\log\frac{\exp(s(v_i, u_i)/\tau)}{\sum_{j=1}^{N}\exp(s(v_i, u_j)/\tau)},\end{aligned} \tag{14}$$

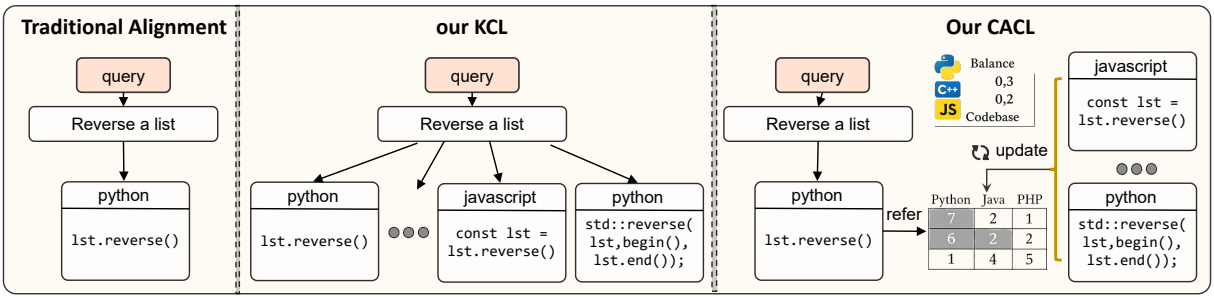

Figure 2: Comparison of different alignment strategies for retrieving code snippets based on the natural language query "Reverse a list". (a) Traditional alignment typically finds corresponding code in only a single programming language (e.g., Python). (b) Our proposed KCL (1-to-K Contrastive Learning) method can use embedding relevant code from multiple programming languages (e.g., Python, Javascript), but may be limited by the distribution of languages in the corpus and resource limit. (c) Our further proposed CACL (Query-Code Co-Anchor Contrastive Learning) method, by incorporating balance weights for the multilingual codebase and an iterative update mechanism, and referring to inter-language similarities.

where u_i and v_i represent input embeddings from different modalities or languages. The three components are defined as follows:

- Code-Primary Query Alignment ($\mathcal{L}_{cacl}^{cq_1}$): $u_i = f_\theta(c_i)$ represents code embeddings, and $v_i = g_\phi(q_{i1})$ represents primary language query embeddings.

- Code-Multilingual Query Alignment ($\mathcal{L}_{cacl}^{cq_{rand}}$): $u_i = f_\theta(c_i)$ represents code embeddings, and $v_i = g_\phi(q_{iq_i})$ represents query embeddings in a randomly selected language.

- Primary Query-Multilingual Query Alignment ($\mathcal{L}_{cacl}^{q_1 q_{rand}}$): $u_i = g_\phi(q_{i1})$ represents primary language query embeddings, and $v_i = g_\phi(q_{iq_i})$ represents query embeddings in a randomly selected language.

The effectiveness of query-code CACL can be explained from two perspectives:

- **Modality Alignment:** From the perspective of modality alignment, the loss function $\mathcal{L}_{cacl}^{q_1 q_{rand}}$ in CACL brings embeddings of the primary language query and other languages closer, reducing the distance between the code embeddings $g_\phi(c_1), g_\phi(c_2)$ and the mean $\frac{1}{2}g_\phi(c_1) + g_\phi(c_2)$ and minimizing the deviation in the modality alignment direction of code and query.

- **Data Distribution Error:** From the perspective of data distribution error $\sum_{(c,q)} ||p(c,q) - p'_e(c,q)||$ in an adapted Eq. (5), CACL's loss functions $\mathcal{L}_{cacl}^{cq_1}$, $\mathcal{L}_{cacl}^{cq_{rand}}$ ensures that the model learns more pairs of code-queries in an epoch. The query in them also contains a large percentage of high-quality primary language text. It makes the data distribution in CACL closer to the original one, and reduces the weight error of the model.

Note that in CACL, the number of queries used for training in each epoch does not increase with the number of languages, which effectively reduces both GPU memory and time overhead in Multilingual Query-Code scenarios with a large number of languages. Our experimental results aim to illustrate that CACL approximates the training time and explicit memory overhead of existing Multilingual Query-Code schemes, yet achieves recall and consistency metrics close to those of 1-to-K comparative learning.

5 Experiment

5.1 Experiment Setup

Dataset To evaluate the performance of our model in a multilingual environment, we used the CodeSearchNet dataset (Husain et al., 2019), a widely used dataset that collects code snippets and their related queries in six different programming languages (including Go, Python, Java, JavaScript, Ruby and PHP) from GitHub. More information about CodeSearchNet can be found in Appendix B and Table 4.

At the same time, in order to better support multilingual retrieval tasks, we further processed the dataset. Considering that each query in the CodeSearchNet dataset usually corresponds to the code

Table 1: Performance Comparison of Different Frameworks and Models across Different Programming Languages

Framework	Model	Language Performance (MRR)							Inconsistency Score (RDM)
		Ruby	JavaScript	Go	Python	Java	PHP	Overall	
Multilingual	RoBERTa(code)	46.0	46.3	82.1	54.7	56.1	52.3	56.2	0.48
	CodeBERT	50.2	50.1	83.8	59.2	59.9	55.6	59.8	0.47
	GraphCodeBERT	51.7	51.4	84.6	62.6	61.4	58.6	61.7	0.46
	UnixCoder	57.1	56.8	86.4	65.3	65.6	60.3	65.3	0.44
Distill	RoBERTa(code)	45.3	46.5	81.6	56.0	57.7	53.5	56.8	0.45
	CodeBERT	49.8	49.1	82.2	60.9	61.8	57.1	60.2	0.44
	GraphCodeBERT	51.3	50.3	84.0	64.2	63.6	60.3	62.3	0.43
	UnixCoder	58.2	58.9	88.2	67.0	67.3	61.9	65.4	0.41
CONSIDER	RoBERTa(code)	52.6	53.4	87.4	60.2	62.7	59.1	62.6	0.42
	CodeBERT	56.2	57.0	89.0	64.6	66.5	62.3	65.9	0.41
	GraphCodeBERT	57.8	57.6	89.5	66.1	67.3	63.1	66.9	0.40
	UnixCoder	61.6	60.9	90.2	69.7	69.9	65.0	69.6	0.38
MD3R-KCL	RoBERTa(code)	**70.5** (+17.9)	**69.8** (+16.4)	**93.1** (+5.7)	**75.2** (+15.0)	**77.4** (+14.7)	**73.0** (+13.9)	**76.5** (+13.9)	0.22
	CodeBERT	**74.1** (+17.9)	**73.5** (+16.5)	**94.0** (+5.0)	**78.5** (+13.9)	**80.1** (+13.6)	**76.3** (+14.0)	**79.4** (+13.5)	0.21
	GraphCodeBERT	**75.8** (+18.0)	**75.0** (+17.4)	**94.7** (+5.2)	**80.1** (+14.0)	**81.5** (+14.2)	**77.9** (+14.8)	**80.8** (+13.9)	0.20
	UnixCoder	**81.9** (+20.3)	**79.9** (+19.0)	**97.4** (+7.2)	**82.6** (+12.9)	**86.1** (+16.2)	**89.8** (+24.8)	**86.3** (+16.7)	0.18
MD3R-CACL	RoBERTa(code)	**69.0** (+16.4)	**68.5** (+15.1)	**92.0** (+4.6)	**73.8** (+13.6)	**76.0** (+13.3)	**71.5** (+12.4)	**75.0** (+12.4)	0.27
	CodeBERT	**72.8** (+16.6)	**72.2** (+15.2)	**92.8** (+3.8)	**77.2** (+12.6)	**78.9** (+12.4)	**75.0** (+12.7)	**78.1** (+12.2)	0.26
	GraphCodeBERT	**74.5** (+16.7)	**73.8** (+16.2)	**93.5** (+4.0)	**79.0** (+12.9)	**80.4** (+13.1)	**76.3** (+13.2)	**79.5** (+12.6)	0.25
	UnixCoder	**80.5** (+18.9)	**78.5** (+17.6)	**96.0** (+5.8)	**81.2** (+11.5)	**84.7** (+14.8)	**87.5** (+22.5)	**85.0** (+15.4)	0.23

of only one specific programming language, we performed an additional translation: we translated the query content of each query (or its description as a code label) into all six languages covered by the dataset with the help of ChatGPT API. In this way, each query has six different language versions, which greatly improves the applicability of the dataset for cross-language code retrieval and matching.

Evaluation Task Evaluation Tasks. Model performance was assessed in both monolingual and multilingual scenarios. In the monolingual scenario, evaluations were conducted on the specific test set for each individual language. To replicate a real-world multilingual context, the test sets from all languages were combined to form a single test set for the multilingual scenario evaluation. The mean reciprocal rank (MRR) (Hull) and Recall@K was used as the evaluation metric for all models. Meanwhile, to measure the dispersion or consistency of retrieval result rankings across different code language representations, a Rank Dispersion Metric (RDM) is proposed based on Mean Variance (Batur and Choobineh, 2010):

$$\mu_{N_i} = \frac{1}{N \cdot K} \sum_{k'=1}^{K} R_{il'} \tag{15}$$

$$RDM = \frac{1}{N \cdot K} \sum_{i=1}^{N} \sum_{k=1}^{K} (R_{il} - \mu_{R_i})^2 \tag{16}$$

In these formulas, N is the total number of data items, and K is the total number of code languages. R_{ik} refers to the rank of the i-th item under the k-th code language, while μ_{R_i} is the mean rank of the i-th item across all code languages. RDM calculates the mean of the squared differences between each item's rank and its mean rank. A lower RDM value indicates more consistent retrieval rankings across different code languages.

Comparision Method We evaluates the proposed framework, named MD3R, by contrasting it with some established approaches for training multilingual code retrieval models. These methods are:

- **Multilingual Training:** This technique involves pooling training data from various programming languages and subsequently employing contrastive learning algorithms during the training process.
- **Knowledge Distillation:** This approach, detailed in a prior study (Li et al., 2022), trains individual monolingual "teacher" models for each programming language. Following this, multilingual "student" models are trained within a multilingual setting, guided by the teacher models.
- **CONSIDER:** The CommONalities and SpecIalties Driven Multilingual CodE Retrieval Framework, proposed in (Li et al., 2024), addresses data scarcity and language confusion in multilingual code retrieval. It enhances language representations by modeling commonalities and extracts specific features

Table 2: Experimental Evaluation Results Summary

Category	MRR	Recall@1	Recall@5	Recall@10	Recall@20	Recall@50	Recall@100
Overall	0.8630	0.8030	0.9357	0.9603	0.9747	0.9873	0.9923
go	0.9736	0.9580	0.9920	0.9960	0.9980	1.0000	1.0000
java	0.8612	0.7920	0.9440	0.9680	0.9800	0.9920	0.9940
javascript	0.7990	0.7280	0.8820	0.9180	0.9460	0.9760	0.9840
php	0.8983	0.8400	0.9680	0.9900	0.9960	0.9980	0.9980
python	0.8265	0.7580	0.9140	0.9460	0.9660	0.9800	0.9880
ruby	0.8193	0.7420	0.9140	0.9440	0.9620	0.9780	0.9900

using a novel negative sampling algorithm, demonstrating benefits in both multilingual and monolingual scenarios.

Overall Result Our experimental evaluations highlight the significant capabilities of the MD3R framework. The comparative analysis in Table 1 reveals that MD3R, encompassing its KCL and CACL variants, consistently surpasses existing multilingual training methods, knowledge distillation (Distill), and the CONSIDER framework across various base models and all six programming languages, both in retrieval performance and in achieving substantially better cross-lingual consistency via lower Rank Dispersion Metric (RDM) scores; the MD3R-KCL strategy with UnixCoder notably achieved an overall performance of 86.3 and an RDM of 0.18 in these comparisons. Complementing these findings, Table 2 quantifies the retrieval efficacy of a prominent MD3R configuration, demonstrating an overall Mean Reciprocal Rank (MRR) of 0.8630, a Recall@1 of 0.8030, and a Recall@100 of 0.9923, thereby illustrating the framework's capacity for both precise and comprehensive code retrieval.

5.2 Ablation Study

To understand the contribution of different components within the proposed MD3R framework, an ablation study was conducted. This study systematically evaluates the impact of key elements, particularly the tailored contrastive learning strategies KCL and CACL, and the individual loss components within CACL, on the overall multilingual query-code retrieval performance. The evaluation metric used is the Overall Mean Reciprocal Rank (MRR).

Table 3 presents the results of this ablation study, comparing the performance of a baseline model, the full MD3R framework with both KCL and CACL, and variations of the MD3R-CACL model where specific loss components are removed or isolated.

Table 3: Ablation Study Results (Overall MRR %)

Model/Component	Overall MRR (%)
Baseline (Traditional Multilingual Training)	65.0
MD3R-KCL (Full Model)	86.3
MD3R-CACL (Full Model)	85.0
MD3R-CACL without $\mathcal{L}_{cacl}^{cq_1}$	81.5
MD3R-CACL without $\mathcal{L}_{cacl}^{cq_{rand}}$	82.0
MD3R-CACL without $\mathcal{L}_{cacl}^{q_1 q_{rand}}$	80.5
MD3R-CACL with only $\mathcal{L}_{cacl}^{cq_1}$	72.0
MD3R-CACL with only $\mathcal{L}_{cacl}^{cq_{rand}}$	73.5
MD3R-CACL with only $\mathcal{L}_{cacl}^{q_1 q_{rand}}$	68.0

The full MD3R-KCL (86.3% Overall MRR) and MD3R-CACL (85.0%) models substantially outperform the baseline (65.0%). Critically, ablating individual components of MD3R-CACL underscores their necessity. Removing the $\mathcal{L}_{cacl}^{cq_1}$, $\mathcal{L}_{cacl}^{cq_{rand}}$, or $\mathcal{L}_{cacl}^{q_1 q_{rand}}$ loss terms reduces MRR to 81.5%, 82.0%, and 80.5%, respectively. Moreover, relying solely on any single CACL loss component results in significantly degraded performance: $\mathcal{L}_{cacl}^{cq_1}$ alone yields 72.0% MRR, $\mathcal{L}_{cacl}^{cq_{rand}}$ alone yields 73.5%, and $\mathcal{L}_{cacl}^{q_1 q_{rand}}$ alone yields 68.0%. This analysis confirms that all evaluated components, particularly the synergistic CACL loss functions, are integral to MD3R's effectiveness in multilingual query-code retrieval.

6 Conclusion

To address the data inconsistency issue in Multilingual Code Retrieval (MLCR), this paper proposes the MD3R framework. Through theoretical analysis and meticulously designed contrastive learning strategies (CACL and KCL), MD3R effectively alleviates data distribution bias and significantly enhances cross-lingual embedding alignment and retrieval consistency. Experiments on CodeSearchNet demonstrate that MD3R improves recall and consistency across six mainstream programming languages, with the KCL strategy showing particularly outstanding performance, achieving state-of-the-art (SOTA) results, providing an innovative solution for MLCR inconsistency issues.

Limitation

Although our MD3R framework and its constituent KCL and CACL strategies demonstrated significant effectiveness in experiments, we also recognize certain **limitations**:

- **Computational Overhead of the KCL Method**: As discussed in Section 4.1 of our paper, the 1-to-K Contrastive Learning (KCL) strategy, while highly effective, can increase GPU memory usage and training time. This overhead is amplified, particularly when supporting a larger number of programming languages.
- **KCL Performance Constraints**: The performance of KCL may be limited by the distribution of languages within the corpus and by available computational resources.
- **CACL as a Trade-off Solution**: To alleviate some of the computational overhead associated with KCL, we proposed the Co-Anchor Contrastive Learning (CACL) strategy (Section 4.2 of our paper). CACL aims to improve consistency and recall with a smaller increase in computational overhead, though its performance gains might be slightly less than those of KCL.

We believe that future research can explore these limitations, for instance, by investigating more efficient contrastive learning methods or by optimizing computational resource allocation for specific application scenarios. Despite these limitations, our work provides an innovative solution to the inconsistency problem in MLCR and has achieved significant experimental results.

References

Loubna Ben Allal, Raymond Li, Denis Kocetkov, Chenghao Mou, Christopher Akiki, Carlos Munoz Ferrandis, Niklas Muennighoff, Mayank Mishra, Alex Gu, Manan Dey, Logesh Kumar Umapathi, Carolyn Jane Anderson, Yangtian Zi, Joel Lamy Poirier, Hailey Schoelkopf, Sergey Troshin, Dmitry Abulkhanov, Manuel Romero, Michael Lappert, and 22 others. 2023. SantaCoder: Don't reach for the stars! *Preprint*, arXiv:2301.03988.

Shushan Arakelyan, Anna Hakhverdyan, Miltiadis Allamanis, Luis Garcia, Christophe Hauser, and Xiang Ren. 2022. Ns3: Neuro-symbolic semantic code search. *Advances in Neural Information Processing Systems*, 35:10476–10491.

Demet Batur and F Fred Choobineh. 2010. Mean-variance based ranking and selection. In *Proceedings of the 2010 winter simulation conference*, pages 1160–1166. IEEE.

Louis Bethune, Thomas Massena, Thibaut Boissin, Yannick Prudent, Corentin Friedrich, Franck Mamalet, Aurelien Bellet, Mathieu Serrurier, and David Vigouroux. 2024. DP-SGD Without Clipping: The Lipschitz Neural Network Way. *Preprint*, arXiv:2305.16202.

Qingying Chen and Minghui Zhou. 2018. A neural framework for retrieval and summarization of source code. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 826–831.

Yanmin Dong, Zhenya Huang, Zheng Zhang, Guanhao Zhao, Likang Wu, Hongke Zhao, Binbin Jin, and Qi Liu. 2025. Enhancing code search intent with programming context exploration. In *Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining*, pages 596–605.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and 1 others. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*.

David A Hull. Xerox TREC-8 Question Answering Track Report.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*.

Jinhan Kim, Sanghoon Lee, Seung-won Hwang, and Sunghun Kim. 2010. Towards an intelligent code search engine. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pages 1358–1363.

Chris Lattner and Vikram Adve. 2004. Llvm: A compilation framework for lifelong program analysis & transformation. In *International symposium on code generation and optimization, 2004. CGO 2004.*, pages 75–86. IEEE.

Otávio AL Lemos, Adriano C de Paula, Felipe C Zanichelli, and Cristina V Lopes. 2014. Thesaurus-based automatic query expansion for interface-driven code search. In *Proceedings of the 11th working conference on mining software repositories*, pages 212–221.

Rui Li, Liyang He, Qi Liu, Yuze Zhao, Zheng Zhang, Zhenya Huang, Yu Su, and Shijin Wang. 2024. Consider: Commonalities and specialties driven multilingual code retrieval framework. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 8679–8687.

Wen Li, Junfei Xu, and Qi Chen. 2022. Knowledge Distillation-Based Multilingual Code Retrieval. *Algorithms*, 15(1):25.

Fei Lv, Hongyu Zhang, Jian-guang Lou, Shaowei Wang, Dongmei Zhang, and Jianjun Zhao. 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)*, pages 260–270. IEEE.

Joao Monteiro, Torsten Scholak, Virendra Mehta, David Vazquez, and Christopher Pal. 2023. Multilingual Code Retrieval Without Paired Data: New Datasets and Benchmarks.

Ruchir Puri, David S Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, and 1 others. 2021. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. *arXiv preprint arXiv:2105.12655*.

Shangwen Wang, Mingyang Geng, Bo Lin, Zhensu Sun, Ming Wen, Yepang Liu, Li Li, Tegawendé F. Bissyandé, and Xiaoguang Mao. 2023. Natural Language to Code: How Far Are We? In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 375–387, San Francisco CA USA. ACM.

Ziyu Yao, Jayavardhan Reddy Peddamail, and Huan Sun. 2019. Coacor: Code annotation for code retrieval with reinforcement learning. In *The world wide web conference*, pages 2203–2214.

Wei Ye, Rui Xie, Jinglei Zhang, Tianxiang Hu, Xiaoyin Wang, and Shikun Zhang. 2020. Leveraging code generation to improve code retrieval and summarization via dual learning. In *Proceedings of The Web Conference 2020*, pages 2309–2319.

Feng Zhang, Haoran Niu, Iman Keivanloo, and Ying Zou. 2017. Expanding queries for code search using semantically related api class-names. *IEEE Transactions on Software Engineering*, 44(11):1070–1082.

Maksim Zubkov, Egor Spirin, Egor Bogomolov, and Timofey Bryksin. 2022. Evaluation of contrastive learning with various code representations for code clone detection. *arXiv preprint arXiv:2206.08726*.

A Proof of Weight Error Upper Bound

We analyze the upper bound on the weighting error heuristically based on the stochastic gradient descent (SGD) optimization algorithm. The following is a detailed theoretical proof of the upper bound on the weighting error in Eq. (7).

$Proof$. Based on the definition of the SGD optimization algorithm, we have:

$$\begin{aligned}\mathbf{w}_{eT} &= \mathbf{w}_{eT-1} - \eta \sum_{(q,c)} p(q,c) \nabla_{\mathbf{w}_{eT-1}} E_{(q,c)}[\log p(q,c)],\\ \mathbf{w}'_{eT} &= \mathbf{w}'_{eT-1} - \eta \sum_{(q,c)} p'_e(q,c) \nabla_{\mathbf{w}'_{eT-1}} E_{(q,c)}[\log p(q,c)].\end{aligned} \tag{17}$$

$$\begin{aligned}
&||\mathbf{w}_{eT} - \mathbf{w}'_{eT}||\\
&= ||\mathbf{w}_{eT-1} - \eta \sum_{(q,c)} p(q,c) \nabla_{\mathbf{w}_{eT-1}} E_{(q,c)}[\log p(q,c)]\\
&\quad - \mathbf{w}'_{eT-1} + \eta \sum_{(q,c)} p'_e(q,c)\\
&\qquad \nabla_{\mathbf{w}'_{eT-1}} E_{(q,c)}[\log p(q,c)]\Big\|\\
&\leq^1 ||\mathbf{w}_{eT-1} - \mathbf{w}'_{eT-1}||\\
&\quad + \eta || \sum_{(q,c)} p'_e(q,c) \nabla_{\mathbf{w}'_{eT-1}} E_{(q,c)}[\log p(q,c)]\\
&\quad - \sum_{(q,c)} p(q,c) \nabla_{\mathbf{w}_{eT-1}} E_{(q,c)}[\log p(q,c)]||\\
&= ||\mathbf{w}_{eT-1} - \mathbf{w}'_{eT-1}||\\
&\quad + \eta || \sum_{(q,c)} p'_e(q,c) \nabla_{\mathbf{w}'_{eT-1}} E_{(q,c)}[\log p(q,c)]\\
&\quad - \sum_{(q,c)} p'_e(q,c) \nabla_{\mathbf{w}_{eT-1}} E_{(q,c)}[\log p(q,c)]\\
&\quad + \sum_{(q,c)} p'_e(q,c) \nabla_{\mathbf{w}_{eT-1}} E_{(q,c)}[\log p(q,c)]\\
&\quad - \sum_{(q,c)} p(q,c) \nabla_{\mathbf{w}_{eT-1}} E_{(q,c)}[\log p(q,c)]||\\
&\leq^2 ||\mathbf{w}_{eT-1} - \mathbf{w}'_{eT-1}||\\
&\quad + \eta || \sum_{(q,c)} p'_e(q,c) (\nabla_{\mathbf{w}'_{eT-1}} E_{(q,c)}[\log p(q,c)]\\
&\quad - \nabla_{\mathbf{w}_{eT-1}} E_{(q,c)}[\log p(q,c)])||\\
&\quad + \eta \Big\| \sum_{(q,c)} (p'_e(q,c) - p(q,c))\\
&\qquad \nabla_{\mathbf{w}_{eT-1}} E_{(q,c)}[\log p(q,c)]\Big\|\\
&\leq^3 (1 + \eta \sum_{(q,c)} p'_e(q,c) \lambda_{(q,c)}) ||\mathbf{w}_{eT-1} - \mathbf{w}'_{eT-1}||\\
&\quad + \eta g_{max}(\mathbf{w}_{eT-1}) \sum_{(q,c)} ||p'_e(q,c) - p(q,c)||.
\end{aligned} \tag{18}$$

The inequality 1 and 2 hold because the Triangle Inequality $|a+b| \leq |a| + |b|$. The inequality 3 holds because

$$g_{max}(\mathbf{w}_{eT-1}) = \max_{(q,c)} ||\nabla_{\mathbf{w}_{eT-1}} E_{(q,c)}[\log p(q,c)]||, \tag{19}$$

and we assume that $\nabla_{\mathbf{w}'_{eT-1}} E_{(q,c)}[\log p(q,c)]$ and $\nabla_{\mathbf{w}_{eT-1}} E_{(q,c)}[\log p(q,c)]$ are $\lambda_{(q,c)}$-Lipschitz. Gradient trimming can be used in the code implementation to a certain extent to reduce the gradient

change in the training process, indirectly reduce the excessive growth of Lipschitz constant, as far as possible to meet the Lipschitz continuity condition.

Based on Eq. (18), let

$$a_e = (1 + \eta \sum_{(q,c)} p'_e(q,c)\lambda_{(q,c)}), \tag{20}$$

we have

$$\begin{aligned}
&||\mathbf{w}_{eT} - \mathbf{w}'_{eT}|| \\
&\quad + \eta g_{max}(\mathbf{w}_{eT-1}) \sum_{(q,c)} ||p'_e(q,c) - p(q,c)|| \\
\leq & a^2 ||\mathbf{w}_{eT-2} - \mathbf{w}'_{eT-2}|| \\
&\quad + \eta \sum_{(q,c)} ||p'_e(q,c) - p(q,c)|| (g_{max}(\mathbf{w}_{eT-1}) \\
&\quad + a g_{max}(\mathbf{w}_{eT-2})) \\
\leq & a^T ||\mathbf{w}_{(e-1)T} - \mathbf{w}'_{(e-1)T}|| \\
&\quad + \eta \sum_{(q,c)} ||p'_e(q,c) - p(q,c)|| (\sum_{j=0}^{T-1} a^j g_{max}(\mathbf{w}_{eT-1-j})) \\
\leq & \eta \sum_{i}^{e} \sum_{(q,c)} ||p'_i(q,c) - p(q,c)|| \\
&\quad \cdot (\sum_{j=(i-1)T}^{iT-1} a_i^j g_{max}(\mathbf{w}_{iT-1-j})).
\end{aligned} \tag{21}$$

Thus Eq. (7) is proved successfully.

B CodeSearchNet Dataset

The distribution of data across CodeSearchNet languages for training, validation, testing, and the overall codebase is detailed in Table 4. It is considered to be the largest and most widely used resource for measuring code retrieval performance.

Table 4: CodeSearchNet Dataset Statistics

Language	Training	Validation	Test	Codebase
Ruby	2.5K	1.4K	1.2K	4.4K
JavaScript	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

C Hypothesis

Our central hypothesis posits that the inconsistencies observed in multilingual code retrieval predominantly arise from systematic discrepancies in the data distribution during the training phase. While our work offers a theoretical foundation for this claim, particularly through the analysis of modality alignment direction errors and the derivation of upper bounds on weight errors, we acknowledge that the current theoretical framework predominantly addresses this specific dimension of data distribution. Nevertheless, other potential sources of inconsistency, although not explored with the same depth in this work, merit consideration. These include the inherent characteristics of model architectures, the incorporation of training paradigms beyond contrastive learning, and the distributional differences inherent in the code across various programming languages. In future research, we intend to expand upon these factors and further investigate their contribution to inconsistencies.

D Related Work about Code Generation Methods

While code generation is a distinct task, techniques from this domain have been adapted as auxiliary objectives to improve code retrieval. The underlying idea is that a model capable of generating code from a query, or vice-versa, develops a deeper understanding of the query-code relationship. For instance, training models to generate natural language summaries or documentation for code snippets helps the model learn better code representations, which in turn benefits retrieval (Yao et al., 2019; Ye et al., 2020). Similarly, training a model to generate code snippets from natural language queries can serve as a powerful auxiliary task. This forces the model to learn a fine-grained mapping between natural language intent and code structures, which can improve its ability to retrieve relevant existing code (Husain et al., 2019). By leveraging these generation-oriented tasks, retrieval models can gain a more nuanced understanding of code semantics and query intent, leading to improved performance in finding relevant code.