CoIR: A Comprehensive Benchmark for Code Information Retrieval Models

Xiangyang Li* **Kuicai Dong*** **Yi Quan Lee*** **Wei Xia**
Hao Zhang† **Xinyi Dai** **Yasheng Wang** **Ruiming Tang†**
HUAWEI NOAH'S ARK LAB
{lixiangyang34, kuicai.dong, lee.yi.quan, xiawei24,
zhang.hao3, daixinyi5,wangyasheng, tangruiming}@huawei.com

Abstract

Despite the substantial success of Information Retrieval (IR) in various NLP tasks, most IR systems predominantly handle queries and corpora in natural language, neglecting the domain of code retrieval. Code retrieval is critically important yet remains under-explored, with existing methods and benchmarks inadequately representing the diversity of code in various domains and tasks. Moreover, many models have begun to overfit existing leaderboards, limiting their generalizability and real-world applicability. Addressing this gap, we introduce **CoIR** (**Co**de **I**nformation **R**etrieval Benchmark), a robust and comprehensive benchmark specifically designed to evaluate code retrieval capabilities. CoIR consists of **ten** meticulously curated code datasets, all of which have undergone thorough manual inspection and processing. These datasets cover **eight** distinct retrieval tasks across **seven** diverse domains, ensuring a broad and rigorous assessment of code retrieval performance. We first discuss the construction of CoIR and its diverse dataset composition. Further, we evaluate ten widely used retrieval models using CoIR, uncovering significant difficulties in performing code retrieval tasks even with state-of-the-art systems. To ensure seamless integration, CoIR is released as a user-friendly Python framework, aligned with the data schema of MTEB and BEIR for consistent cross-benchmark evaluation. Through CoIR, we aim to invigorate research in the code retrieval domain, providing a versatile benchmarking tool that encourages further development and exploration of code retrieval systems[1].

1 Introduction

Information retrieval (IR) aims to retrieve relevant information from a large scale corpus. The advances of pretrained Transformers (Vaswani et al., 2017) like BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019) enable IR systems (Wang et al., 2022; Chen et al., 2024; Li et al., 2023b; Dong et al., 2025a) to leverage rich semantic embeddings to interpret and fulfill natural language search queries. Specifically, IR have successfully improve a wide range of Natural Language Processing (NLP) tasks such as Question Answering (QA) (Kolomiyets and Moens, 2011), Retrieval-Augmented Generation (RAG) (Liu et al., 2024; Dong et al., 2024, 2025b; Zhang et al., 2025; Jia et al., 2024), Information Extraction (Ziembinski, 2015; Sarhan, 2023), Text Summarization (Mahalakshmi and Fatima, 2022), Recommender System (Li et al., 2022, 2023a; Lin et al., 2025; Wang et al., 2024a) and etc. Although proven to be effective in text retrieval, standard IR methods often fall short in code retrieval (Husain et al., 2019).

Benchmark	Domain	#PL	Retrieval Tasks	Eval Package
CoSQA	Web Query	1	Text-to-Code	×
CodeSearchNet	GitHub Functions	6	Text-to-Code	×
CodeRAG-Bench	Contest, Issue Fixing, StackOverflow, GitHub Functions	1	Text-to-Code Hybrid Code	×
XCodeEval	Contest	17	Text-to-Code Code-to-Code	×
CoIR (**Ours**)	GitHub Functions, Web Query, Database, Contest, Deep Learning, StackOverflow, Code Instruction	14	Text-to-Code Code-to-Code Code-to-Text Hybrid Code	✓

Table 1: Comparison between CoIR and other code retrieval benchmarks. #PL is the number of main programming languages used.

Unlike standard text, code is semi-structured and inherently logical, consisting of syntactic rules and semantic information that require specific parsing and understanding. Such distinctive nature requires the system to adapt and interpret code

*These authors contributed equally to this work.
†Corresponding authors.
[1] https://github.com/CoIR-team/coir

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 22074–22091
July 27 - August 1, 2025

format accurately. Recognizing the importance of code data, pioneering works such as CodeBERT (Feng et al., 2020), CodeGPT (Lu et al., 2021) and UniXcoder (Guo et al., 2022) have conducted pre-training specifically on code corpora.

Code information retrieval is a critical component in accelerating development processes and improving code quality. Efficient code retrieval helps developers quickly find not only relevant code snippets, but also related information like code explanations, bug analyses, code summaries, and similar code instances. Commercial products have recently integrated tools for code retrieval, such as VS Code (Sole, 2019) and GitHub Code Search. Moreover, code-RAG systems (Zhang et al., 2023b,a; Choi et al., 2023; Su et al., 2024) have effectively leveraged on code retrieval to minimize hallucinations (i.e., errors in generated code) by Large Language Models (LLMs), thereby ensuring more accurate and reliable outputs.

Due to the importance of code retrieval, benchmarks (see Table 1) such as CodeSearchNet (Husain et al., 2019), CosQA (Huang et al., 2021), and XcodeEval (Khan et al., 2023), have been proposed to evaluate the code retrieval effectiveness. Despite these efforts, there remain three principal limitations: **(1)** *Current benchmarks focus on a limited number of code retrieval tasks and have been extensively overfitted by many existing models*. Commonly, these involve using a textual query to search for corresponding code snippets. However, the practical needs of code retrieval are far more diverse. In real-world scenarios, queries and retrieved corpus can involve both text and code. One might input a code snippet coupled with bug information, and seek detailed explanations, summaries, or even fixed code as output. Existing benchmarks do not adequately cater to such complex and varied query types, limiting the scope of their applicability and the robustness of the models tested. **(2)** *there is a noticeable lack of diversity in data domains*. For example, CodeSearchNet exclusively extracts code and code-comment pairs from GitHub, which represents a specific format of open-source projects. Similarly, XcodeEval focuses only on coding related to contest challenges, which may be overly specialized. Such narrow focus is not suitable for comprehensive evaluation in broader coding contexts. **(3)** *there is no standard evaluation framework for code retrieval*, which complicates the comparison and development of methods in this field. CodeSearchNet, CosQA, and XcodeEval employ various types of evaluation metrics tailored to their specific tasks and formats leading to potential inconsistencies in measuring model performance across different benchmarks.

To address the limitations of existing code retrieval benchmarks, we introduce **CoIR** (**Co**de **I**nformation **R**etrieval Benchmark), a more comprehensive and versatile benchmark. As summarized in Table 1, CoIR surpasses current benchmarks by offering broader domain coverage, more diverse retrieval tasks, and a standardized evaluation framework. It includes 10 datasets (8 existing and 2 newly-curated) and supports 4 primary retrieval tasks: (1) Text-to-Code, (2) Code-to-Code, (3) Code-to-Text, and (4) Hybrid Code Retrieval, with further breakdown into 8 sub-tasks. The datasets vary in size, ranging from 1K to 1M documents, with token counts ranging from 37 to 4.4K for queries and 113 to 1.5K for corpus.

We evaluated 10 popular retrieval models on CoIR, revealing that even state-of-the-art models perform suboptimally, highlighting the complexity of code retrieval. To simplify evaluation, we also provide a user-friendly Python framework that integrates seamlessly with BEIR and MTEB, allowing easy model evaluation via pip installation. Our key contributions are as follows:

- We present the first benchmark designed to comprehensively evaluate code retrieval. This benchmark, CoIR, integrates 10 datasets and addresses 4 key code retrieval tasks. To achieve this, we manually collected and curated existing datasets through a rigorous data cleaning and filtering process, while also creating two new datasets.
- Our evaluation of 10 popular retrieval systems reveals that even state-of-the-art models struggle in code retrieval. Through experimental analysis, we demonstrate that LLM-based retrieval models have the potential to pave the way for new directions in code and text retrieval. Additionally, our findings indicate that many existing models have already overfitted to current leaderboards, highlighting the necessity for more generalizable and robust retrieval approaches.
- We offer a robust community around CoIR, providing convenient evaluation tools, leaderboards, and other resources that facilitate the rapid advancement of code retrieval models.

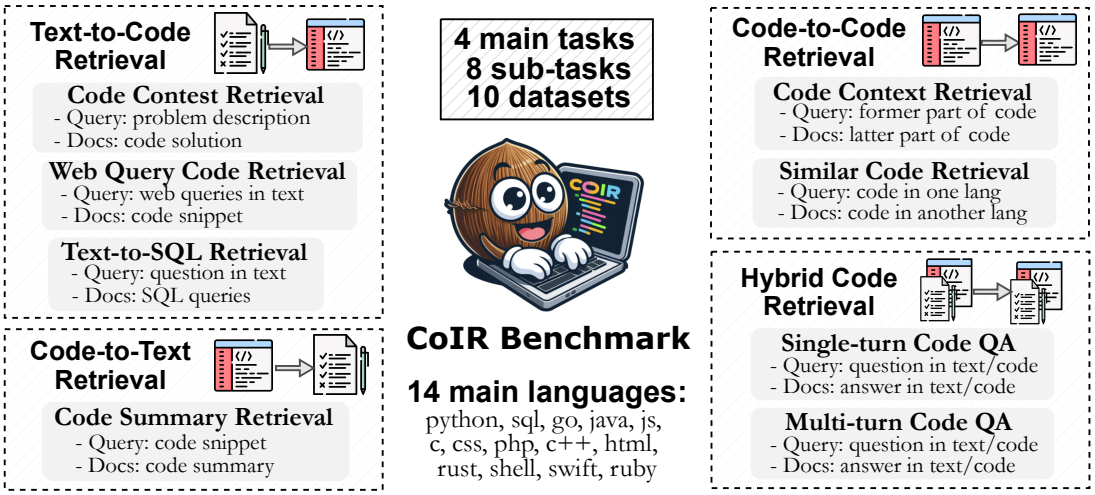

Figure 1: Overview of CoIR benchmark.

2 Related Work

Existing Benchmarks. BEIR (Thakur et al., 2021) and MTEB (Muennighoff et al., 2023) have significantly advanced retrieval and embedding models across diverse text-based tasks but lack dedicated evaluations for code retrieval. Existing code benchmarks, such as CodeSearchNet (Husain et al., 2019), CosQA (Huang et al., 2021), and XcodeEval (Khan et al., 2023), suffer from limitations in task diversity, domain coverage, and evaluation consistency. Notably, CodeSearchNet has been extensively used, leading to overfitting in many models. In contrast, CoIR introduces a broader range of code retrieval tasks and a unified evaluation framework for standardized benchmarking.

Retrieval Models. Dense retrievers, which encode text into high-dimensional representations, are central to information retrieval (IR). Key methods include DPR (Karpukhin et al., 2020), Contriever (Izacard et al., 2022), E5 (Wang et al., 2022), GTE (Li et al., 2023b), BGE (Xiao et al., 2023), and BGE-M3 (Bai et al., 2024), with recent advances leveraging LLMs, such as E5-Mistral (Wang et al., 2024b) and OpenAI-Ada-002 .

However, most research focuses on QA datasets like MS-Marco (Nguyen et al., 2016), with limited work on code retrieval. While models like CodeBERT (Feng et al., 2020), Voyage-Code-002, and UniXCoder (Guo et al., 2022) exploit programming language structures, neural retrieval for code remains underexplored. By providing benchmarks for code retrieval , we aim to stimulate innovation and advance state-of-the-art techniques in this area.

3 The CoIR Benchmark

3.1 Desiderata

CoIR provides a one-stop zero-shot evaluation benchmark for diverse code retrieval tasks, ensuring comprehensive assessment through well-curated tasks and datasets. To streamline evaluation, CoIR includes a one-click, end-to-end pipeline with three key features:

(1) **Diversity**: CoIR assesses models across 4 primary tasks, 8 sub-tasks, 10 datasets, and 14 programming languages (Figure 1). (2) **Usability**: Unlike traditional evaluations that require manual coding and result collection, CoIR offers an automated pipeline for both open-source and proprietary models, supporting metrics such as nDCG, precision, recall, and MAP. Results are stored in `JSON` format for easy access. (3) **Overfitting Mitigation**: Many models overfit to benchmarks like CodeSearchNet, leading to inflated performance with limited generalization. CoIR alleviates this by incorporating diverse tasks and datasets, offering a more reliable evaluation. Dataset statistics are in Table 2, with preparation details in B.1.

3.2 Overview of CoIR Tasks

In this section, we present an overview of each task in CoIR. For each task-specific dataset, we manually inspect and filter out instances that lack valid answers, exhibit ambiguity, contain irrelevant information, etc. The rationale for selecting each dataset, along with the detailed manual filtering process, is provided in Appendix B.1.

Main Task	Sub Task	Domain	Dataset	Language	#Query (train/dev/test)	#Corpus	L_{Query}	L_{Corpus}
Text-to-Code Retrieval	Code Contest Retrieval	Code Contest	APPS	py	5k/-/3.8K	9K	1.4K	575
	Web Query to Code Retrieval	Web query	CosQA	py	19k/-/500	21K	37	276
	Text to SQL Retrieval	Database	Synthetic Text2SQL	sql	100k/-/6K	106K	83	127
Code-to-Text Retrieval	Code Summary Retrieval	Github Fuctions	CodeSearchNet	go, java, js php, py, ruby	905k/41k/53K	1M	594	156
Code-to-Code Retrieval	Code Context Retrieval	Github Fuctions	CodeSearchNet -CCR†	go, java, js php, py, ruby	905k/41k/53K	1M	154	113
	Similar Code Retrieval	Deep Learning	CodeTrans Ocean-DL	py	564/72/180	816	1.6K	1.5K
		Contest	CodeTrans Ocean-Contest	c++, py	561/226/446	1K	770	1.5K
Hybrid Code Retrieval	Single-turn Code QA	Stack Overflow	StackOverflow QA†	miscellaneous	13k/3k/2K	20K	1.4K	1.2K
		Code Instruction	CodeFeedBack -ST	html, c, css, sql js, sql, py, shell ruby, rust, swift	125k/-/31K	156K	722	1.5K
	Multi-turn Code QA	Code Instruction	CodeFeeback -MT	miscellaneous	53k/-/13K	66K	4.4K	1.5K

Table 2: **Statistics of datasets** in COIR benchmark. # is the quantity of query/corpus instances. $L_{(\cdot)}$ refers to the average numbers of words per query/corpus. Datasets marked by † are created by us.

3.2.1 Main Task I: Text-to-Code Retrieval

Code Contest Retrieval. Code contest retrieval involves retrieving relevant code solutions for coding problems described in natural language, a challenging task due to the complexity of language and code, as well as the gap between human and machine languages. For this, we use the APPS dataset (Hendrycks et al., 2021), a diverse collection of problems from platforms like Codewars, AtCoder, Kattis, and Codeforces.

Web Query Code Retrieval. Web query code retrieval focuses on retrieving relevant code snippets based on concise web queries, typically just a few words. For this task, we use the CosQA (Huang et al., 2021) dataset, containing 20.6k labeled pairs of textual queries and Python functions.

Text-to-SQL Retrieval. Text-to-SQL is a key task in code generation, requiring models to generate SQL queries from natural language questions. We use the Synthetic Text-to-SQL dataset (Meyer et al., 2024), the largest and most diverse synthetic dataset, with around 106k examples.

3.2.2 Main Task II: Code-to-Text Retrieval

Code Summary Retrieval. Code summary retrieval evaluates a model's ability to use code to retrieve code summaries or annotations. For this task, we employ the CodeSearchNet dataset (Husain et al., 2019), which consists of numerous code functions accompanied by code comments. This dataset spans six programming languages and includes over one million documents, providing a rich source of information for evaluating model performance.

3.2.3 Main Task III: Code-to-Code Retrieval

Code Context Retrieval. Code context retrieval is the task of retrieving the most relevant code segment that completes a given initial segment of code. This task is critical for code completion purposes. Here we modify the original CodeSearchNet dataset (Husain et al., 2019) to better suit our needs. Specifically, for each code snippet or function in CodeSearchNet, we randomly divide the code into two segments: (1) the initial segment serves as our query, and (2) the remaining segment forms the target corpus to be retrieved for this query. The length of each query is uniformly and randomly selected to comprise between 40% and 70% of the total number of characters in the original code.

Similar Code Retrieval. In the task of similar code retrieval, the primary objective is to assess a model's ability to retrieve similar code snippets.

Specifically, given a code snippet in one programming language or deep learning framework, the model needs to retrieve semantically equivalent code in a different language or framework. We utilize the CodeTransOcean dataset (Yan et al., 2023) for this purpose, creating two sub-datasets for similar code retrieval. These sub-datasets are named "CodeTransOcean-DL" and "CodeTransOcean-Contest", with the suffix indicating their respective domains. The "CodeTransOcean-DL" subset contains code written in different deep learning frameworks within the same programming language, such as TensorFlow and Paddlepaddle. Whereas the "CodeTransOcean-Contest" subset includes code written in different programming languages, such as Python and C++, for the same coding contest or competitive programming problem.

3.2.4 Main Task IV: Hybrid Code Retrieval

Single-turn Code Question Answer Retrieval. In single-turn code question-answering (QA), a retrieval model is required to find the corresponding answer for a given natural language question. Both the question and the answer typically consist of a mix of text and code snippets. For this task, we use two code QA datasets: StackOverflow QA [2] and CodeFeedQA (Zheng et al., 2024). The StackOverflow QA is derived from the original StackOverflow dataset by pairing questions with their highest upvoted answers, resulting in $19,931$ pairs. Additionally, we sampled $1,202$ query instances to validate the retrieval model's performance. The CodeFeedQA is a synthesized code instruction dataset generated by LLMs. From this dataset, we sampled 20% of the queries to assess the model's performance.

Multi-turn Code Question Answer. In multi-turn code question-answer retrieval, the retrieval model must effectively utilize the context from multiple dialogue turns to accurately retrieve the answer for the subsequent turn. This task is challenging due to the extensive dialogue context, which can exceed $4,000$ tokens, whereas most current retrieval models are limited to a context length of 512 tokens. Specifically, we employ the CodeFeedback multi-turn question-answer dataset, with dialogues generated by LLMs. Our test set comprises a total of $13,227$ queries and $66,383$ corpus.

[2] https://www.kaggle.com/datasets/stackoverflow/stacksample/data

3.3 Dataset and Diversity Analysis

In addition to the multitude of tasks, the datasets in CoIR also encompass a broad array of programming languages, such as Python, Java, and SQL, each featuring unique attributes. The distribution of these programming languages is long-tailed, as depicted in the bar graph in Figure 2. Despite this, the datasets maintain diversity, originating from varied sources including code contest websites, GitHub repositories, StackOverflow responses, etc. To quantify the diversity of CoIR, we calculated the weighted Jaccard similarity scores on unigram word overlap across all dataset pairs, displayed in a heatmap in Figure 2 (see Appendix C.2 for calculation details). The heatmap reveals generally low Jaccard similarity scores among dataset pairs, with notable exceptions being CodeFeedback Single-Turn (CodeFeedback-ST) and CodeFeedback Multi-Turn (CodeFeedback-MT), which are derived from the same domain. This signifies the challenge of CoIR as a benchmark. For optimal performance, a method must not only excel in major programming languages but also exhibit robust generalization ability across various domains.

3.4 CoIR Evaluation Software

To advance the evaluation of code retrieval capabilities, we introduce a streamlined, user-friendly Python framework for the CoIR benchmark evaluation. This framework is installable via pip using the command `pip install` and features a straightforward script that evaluates model performance across multiple datasets, outputting results in `JSON` format. Meanwhile, CoIR is compatible with several popular open-source frameworks, including HuggingFace and Sentence-Transformers, as well as API-based models such as OpenAI-Ada-002 and Voyage-Code-002. In line with BEIR and MTEB, all datasets have been standardized into a uniform format, facilitating the use of MTEB and BEIR frameworks for evaluating CoIR.

4 Experiment Setup

In this section, we evaluate and analyze the performance of the current state-of-the-art retrieval models on the eight subtasks of the CoIR benchmark. More details can be found in Appendix D.

Benchmarked Models. To comprehensively evaluate the capabilities of various state-of-the-art retrieval models for code retrieval tasks, we select 10 different retrieval models. For sparse retrieval, we utilized BM25 (Robertson et al.,

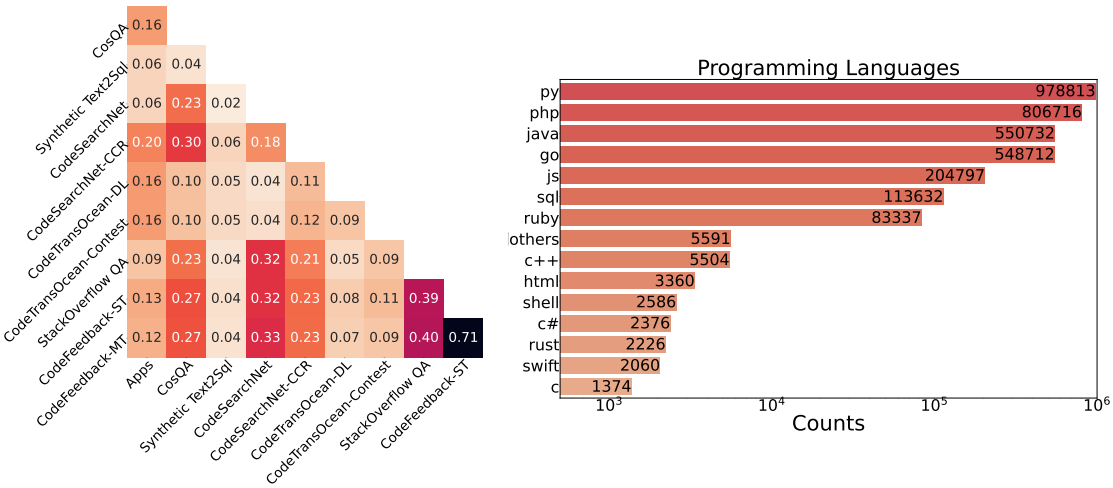

Figure 2: Jaccard similarity and Programming Language Distributions of CoIR.

1994). For open-source dense models, we included E5-Base (Wang et al., 2022), GTE-Base (Li et al., 2023b), BGE-Base (Xiao et al., 2023), Contriever (Izacard et al., 2022), E5-Mistral (Wang et al., 2024b), BGE-M3 (Chen et al., 2024), and UniXcoder (Guo et al., 2022). For proprietary models, we included OpenAI-Ada-002 [3] and Voyage-Code-002 [4]. Notably, both UniXcoder and Voyage-Code-002 have been pre-trained on code data, making them robust baselines for comparison.

Implementation Details. For all open-source dense retrieval models, we set the input length of both queries and the corpus to 512. For proprietary models such as Voyage-Code-002, due to the constraint of tokens per minute (TPM), we set the length of the queries to 256 and the corpus to 512. Following previous work (Wang et al., 2024b), we utilize (i) last token pooling for E5-Mistral, and (ii) mean pooling for other models, to obtain semantic vector representations. Cosine similarity is employed to calculate the similarity between the query and corpus for ranking. Following BEIR (Thakur et al., 2021), we use NDCG@10 to evaluate the performance of retrieval models. For comprehensive evaluation, our scripts also provide metrics such as MAP, Recall, and Precision.

5 Results and Analysis

5.1 Main Results

Table 3 showcases the performance of existing embedding models on the CoIR benchmark. From the table, several conclusions can be drawn:

[3] https://platform.openai.com/docs/guides/embeddings

[4] https://blog.voyageai.com/2024/01/23/voyage-code-2-elevate-your-code-retrieval/

No Single Model Dominates Across All Tasks. Despite Voyage-Code-002 achieving the highest mean score of 56.26 on the benchmark, it does not universally surpass other models in every task. Models like E5-base and E5-Mistral exhibit superior performance in specific retrieval sub-tasks. This highlights the CoIR benchmark's complexity and suggests that different models excel in different facets, catering to varied task requirements.

Demonstration of Benchmark Diversity and Challenge. Models achieving top performance in mainstream text retrieval benchmarks, such as E5-Mistral, do not consistently excel across CoIR sub-tasks. For example, E5-Mistral scores highly in CodeTrans-Contest and StackOverflow QA but shows only median performance in APPS and CosQA for Text-to-Code tasks, and falls behind E5-base in CodeFeedBack-MT. This variation underscores the CoIR benchmark's challenge and the need for adaptive, robust code retrieval capabilities.

Effectiveness of Code-Specific Optimization. Models specialized for code retrieval, like Voyage-Code-002, achieve strong results on CoIR, highlighting the impact of tailored optimization. Voyage-Code-002 excels in both Text-to-Code and Code-to-Text tasks, demonstrating the benefits of domain-specific training.

Figure 3 presents box plots for each model. Voyage-Code-002 achieves the highest average performance but with high variance, suggesting weaker generalization. In contrast, BGE-M3 shows better robustness with the lowest variance despite its moderate performance. This highlights the CoIR dataset's challenge and its value as a benchmark for code retrieval research.

Task (→)	Text-to-Code			Code-to-Text	Code-to-Code			Hybrid Code			Avg
Model (param.) ↓	Apps	CosQA	Synthetic Text2sql	Code SearchNet	Code SN-CCR	CodeTrans -Contest	CodeTrans -DL	StackOver Flow QA	CodeFeedBack -ST	CodeFeedBack -MT	
BM25	0.95	13.96	16.92	26.75	34.69	50.13	8.69	56.80	54.32	34.73	29.79
Contriever (110M)	5.14	14.21	45.46	34.72	35.74	44.16	24.21	66.05	55.11	39.23	36.40
E5-base (110M)	11.52	32.59	52.31	67.99	56.87	62.50	21.87	86.86	74.52	**41.99**	50.90
BGE-Base (110M)	4.05	**32.76**	45.59	69.60	45.56	38.50	21.71	73.55	64.99	31.42	42.77
GTE-Base (110M)	3.24	30.24	46.19	43.35	35.50	33.81	28.80	62.71	55.19	28.48	36.75
UniXcoder (123M)	1.36	25.14	50.45	60.20	58.36	41.82	31.03	44.67	36.02	24.21	37.33
BGE-M3 (567M)	7.37	22.73	48.76	43.23	47.55	47.86	31.16	61.04	49.94	33.46	39.31
E5-Mistral (7B)	21.33	31.27	65.98	54.25	65.27	**82.55**	**33.24**	**91.54**	**72.71**	33.65	55.18
OpenAI-Ada-002	8.70	28.88	58.32	74.21	69.13	53.34	26.04	72.40	47.12	17.74	45.59
Voyage-Code-002	**26.52**	29.79	**69.26**	**81.79**	**73.45**	72.77	27.28	87.68	65.35	28.74	**56.26**

Table 3: NDCG@10 score of various retrievers on CoIR. The best score is marked in boldface.

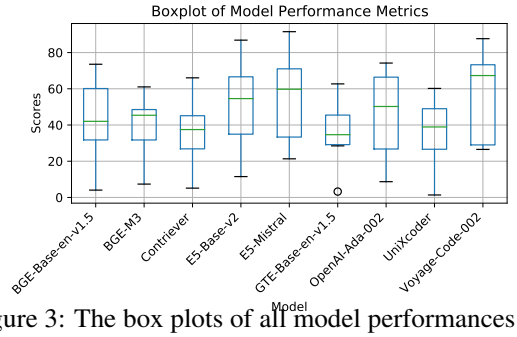

Figure 3: The box plots of all model performances on CoIR benchmark.

CodeFeedBack-ST 156k corpus & 31k queries			Embed Latency	Retrieval Latency	Index
Rank	Model	Dim.	GPU	GPU	Size
9	Contriever	768	7.8ms	38.1µs	0.3G
3	E5-Base	768	7.4ms	38.1µs	0.3G
5	BGE-Base	768	7.6ms	38.1µs	0.3G
8	GTE-Base	768	7.8ms	38.1µs	0.3G
7	UniXcoder	768	7.7ms	38.1µs	0.3G
6	BGE-M3	1024	31.4ms	42.9µs	0.6G
2	E5-Mistral	4096	1840ms	115.5µs	2.3G
4	OpenAI-Ada-002	1536	-	56.8µs	0.9G
1	Voyage-Code-002	1536	-	56.8µs	0.9G

Table 4: The average embedding/retrieval latency per instance using CodeFeedBack-ST. We retain models that have rankings in both BEIR and CoIR.

5.2 Analysis of Retrieval Efficiency

In practical industrial settings, efficiency is also a critical factor for evaluating retrieval model. Factors such as embedding latency for query/corpus, retrieval latency from all embeddings, and the physical memory footprint of the index are critical. In this section, we analyze the efficiency of the current retrieval model using a portion of the CoIR dataset, designated as CodeFeedBack-ST, which comprises 156k corpus and 31k queries. For **embedding latency**, we record the time taken by each model to process a batch of data. This time was then divided by the number of samples in the batch to derive the average embedding time per sample. For **retrieval latency**, we measure the averaged retrieval time of all queries, where each query is tasked with retrieving 1,000 documents. The latency results are presented in Table 4.

Trade-off between Accuracy and Latency. High-performing models like E5-Mistral with excellent accuracy, suffer from noticeable embedding latency. Specifically, E5-Mistral's average embedding latency is 1840ms per sample, significantly higher than other evaluated models. This discrepancy highlights the common trade-off in retrieval systems: higher accuracy often comes at the expense of increased latency.

Index Size Considerations. Index sizes vary significantly across different models. Simpler models such as Contriever, E5-Base, BGE-Base, GTE-Base, and UniXcoder have relatively small index sizes, around 0.3GB. In contrast, more complex models like E5-Mistral, OpenAI-Ada-002, and Voyage-Code-002 have larger index sizes ranging from 0.6GB up to 2.3GB. This suggests a trade-off between accuracy and memory demands. Advanced models with better performance is at the cost of more memory, which could be a limitation in resource-constrained environments.

5.3 Input Length Impact on Code Retrieval

This section examines how input length affects the performance of code retrieval tasks. We analyze results using four datasets: CodeFeedBack-MT, CodeTransOcean-DL, APPS, and StackOverflow QA. Each dataset has an average query and corpus length that exceeds 1,000 words. We utilize two

models, GTE and BGE-M3, both of which have been optimized for long documents and support context length of 8k tokens. We experiment on two settings, with input length capped at 512 and 4,096 respectively. Note that tokens beyond the cap will be truncated. We report the results in Table 5.

Model (input #tokens)	Code FB-MT	Code TO-DL	APPS	Stack OF-QA
GTE (512)	28.48	28.80	3.24	62.71
GTE (4k)	51.32	27.33	5.08	78.63
BGE-M3 (512)	33.46	31.16	7.37	61.04
BGE-M3 (4k)	27.49	32.75	6.80	56.53

Table 5: Effects of different input length.

Impact of Input Length on Model Performance. For GTE model, extending the input length from 512 to 4,096 shows notable improvements in retrieval performance across most datasets. Specifically, retrieval scores in CodeFeedBack-MT and StackOverflow QA increase significantly from 38.20 to 51.32, and from 64.36 to 78.63, respectively. In contrast, the BGE-M3 model shows inconsistent results. CodeTransOcean-DL shows a slight improvement in scores from 31.16 to 32.75, whereas scores for CodeFeedBack-MT fall from 33.46 to 27.49. One possible reason can be that: although BGE-M3 has been optimized for long documents, the significant differences between code data and text data result in a performance degradation as the document length increases.

5.4 Comparison of CoIR and BEIR Rankings

Model	Rank	
	CoIR	BEIR
Contriever	7	7
GTE-Base	6	2
BGE-M3	5	6
BGE-Base	4	3
OpenAI-Ada-002	3	5
E5-Base	2	4
E5-Mistral	1	1

Table 6: Retriever rankings in CoIR and BEIR.

This section evaluates seven retrieval models across CoIR and BEIR, adjusting rankings for consistency while excluding voyage-code-002 and UniXcoder due to their code-specific pretraining. We analyze key ranking shifts and patterns.

Key Insights. E5-Mistral consistently ranks first in both benchmarks, demonstrating robust performance in text and code retrieval. However, GTE-Base drops from 2nd in BEIR to 6th in CoIR, indicating that strong text retrieval does not ensure effective code retrieval. Conversely, E5-Base ranks higher in CoIR than in BEIR, suggesting better adaptation to code-related tasks.

These results underscore the necessity of specialized benchmarks like CoIR. While some models maintain stable rankings, others exhibit notable shifts, highlighting distinct challenges of code retrieval that general text benchmarks may overlook.

5.5 Overfitting in CodeSearchNet and How CoIR Helps

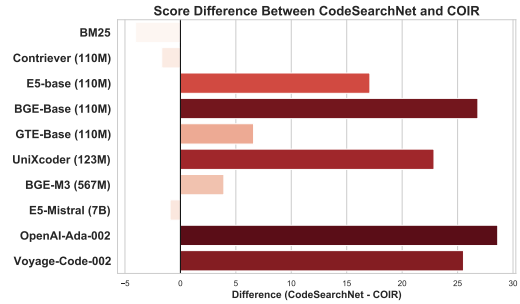

Figure 4: Overfitting in CodeSearchNet: Evidence from Score Differences. The values in the bar chart represent the scores of CodeSearchNet minus the scores of CoIR.

Figure 4 compares the performance of CodeSearchNet and CoIR across various retrieval models. Most models score significantly higher on CodeSearchNet, indicating a strong overfitting tendency. Notably, OpenAI-Ada-002 and Voyage-Code-002 show large performance gaps, suggesting these models may have already overfitted to CodeSearchNet.

In contrast, CoIR provides a more balanced evaluation, mitigating overfitting. Among all models, E5-Mistral (7B) achieves the **smallest performance gap** while maintaining strong scores on both benchmarks, demonstrating that **LLM-based retrieval models can effectively reduce overfitting**. These results highlight CoIR 's role in ensuring a more robust and generalizable benchmark.

6 Conclusion

In this paper, we introduce CoIR, a comprehensive benchmark for code retrieval. CoIR encompasses **4** distinct and **8** fine-grained retrieval tasks, supports **14** programming languages, and integrates **10** diverse datasets with over 2 million code snippets. Furthermore, we evaluate the performance of **10** retrieval models on CoIR, revealing that even state-of-the-art models struggle, underscoring its chal-

lenging nature. Additionally, our analysis suggests that many existing models have overfitted to current leaderboards, highlighting the need for more generalizable and robust retrieval approaches. Moreover, we emphasize the promise of LLM-based retrieval models as a potential direction for future advancements. With CoIR, we aim to foster progress in code retrieval, encouraging researchers to develop more effective and resilient models to benefit the community.

7 Limitations

Even though we cover a wide range of tasks and domains in CoIR, no benchmark is perfect and each set of benchmark has its own limitations. It is crucial to make the limitations explicit in order to better interpret (1) retrieval results on these benchmark datasets and (2) to curate a better benchmark in the future that complements existing benchmarks in the field.

1. **Multilingual Tasks:** Although we aim for a diverse retrieval evaluation benchmark, due to the limited availability of multilingual retrieval datasets for code information retrieval , all datasets covered in the CoIR benchmark are currently English. Future work could include multi- and cross-lingual tasks and models.

2. **Multi-faceted Search:** Due to the existing paradigm that heavily emphasizes on semantic-based information retrieval, our benchmark mainly focuses on queries that aim to benchmark retrieval performance solely based on textual information. However, real-world information needs is often complex and could possibly rely on various corpus meta-data, especially so in code datasets where meta-data could play a huge role. For instance, the versioning of programming language or software libraries could make a huge difference to whether a functioning code snippet is retrieved. In essence incorporating queries that are multi-faceted can more accurately reflect real-world informational needs.

3. **n-ary Match:** With each query corresponding to exactly one ground-truth corpus, we dismiss the real-world informational needs where (1) a single query is could be relevant to multiple copora and (2) the informational needs of a single query can only be satisfied by multiple corpora simultaneously.

   Dedicating a particular section of CoIR for n-nary / list-wise labels for each query would allow us to address how model perform on informational needs that are diverse and further enhance the diversity of tasks of the benchmark.

8 Ethical Considerations

We ensure that the distribution of each dataset complies with the corresponding licenses, all of which are listed below:

- APPS: Provided under "MIT License" for non-commercial research purposes.
- CodeTransOcean: Provided under Apache License 2.0 license.
- CodeSearchNet: Provided under "MIT License" for non-commercial research purposes.
- CoSQA: Provided under "MIT License" for non-commercial research purposes.
- Synthetic Text2sql: Provided under Apache License 2.0 license.
- Code-Feedback: Provided under Apache License 2.0 license.
- CodeFeedback-Filtered-Instruction: Provided under Apache License 2.0 license.
- Stackoverflow QA: Provided under CC-BY-SA 3.0 license.

References

Yang Bai, Anthony M. Colas, Christan Grant, and Daisy Zhe Wang. 2024. M3: A multi-task mixed-objective learning framework for open-domain multi-hop dense sentence retrieval. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy*, pages 10846–10857. ELRA and ICCL.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. BGE m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *CoRR*, abs/2402.03216.

YunSeok Choi, CheolWon Na, Hyojun Kim, and Jee-Hyong Lee. 2023. READSUM: retrieval-augmented adaptive transformer for source code summarization. *IEEE Access*, 11:51155–51165.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Kuicai Dong, Yujing Chang, Xin Deik Goh, Dexun Li, Ruiming Tang, and Yong Liu. 2025a. Mmdocir: Benchmarking multi-modal retrieval for long documents. *Preprint*, arXiv:2501.08828.

Kuicai Dong, Yujing Chang, Shijie Huang, Yasheng Wang, Ruiming Tang, and Yong Liu. 2025b. Benchmarking retrieval-augmented multimomal generation for document question answering. *Preprint*, arXiv:2505.16470.

Kuicai Dong, Derrick Goh Xin Deik, Yi Lee, Hao Zhang, Xiangyang Li, Cong Zhang, and Yong Liu. 2024. Mc-indexing: Effective long document retrieval via multi-view content-aware indexing. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2673–2691.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online. Association for Computational Linguistics.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified cross-modal pre-training for code representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7212–7225, Dublin, Ireland. Association for Computational Linguistics.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring coding challenge competence with apps. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1.

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

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

Sergey Ioffe. 2010. Improved consistent sampling, weighted minhash and l1 sketching. In *2010 IEEE International Conference on Data Mining*, pages 246–255.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised dense information retrieval with contrastive learning. *Trans. Mach. Learn. Res.*, 2022.

Pengyue Jia, Derong Xu, Xiaopeng Li, Zhaocheng Du, Xiangyang Li, Xiangyu Zhao, Yichao Wang, Yuhao Wang, Huifeng Guo, and Ruiming Tang. 2024. Bridging relevance and reasoning: Rationale distillation in retrieval-augmented generation. *arXiv preprint arXiv:2412.08519*.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-scale similarity search with gpus. *IEEE Trans. Big Data*, 7(3):535–547.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Mohammad Abdullah Matin Khan, M. Saiful Bari, Xuan Long Do, Weishi Wang, Md. Rizwan Parvez, and Shafiq R. Joty. 2023. xcodeeval: A large scale multilingual multitask benchmark for code understanding, generation, translation and retrieval. *CoRR*, abs/2303.03004.

Oleksandr Kolomiyets and Marie-Francine Moens. 2011. A survey on question answering technology from an information retrieval perspective. *Inf. Sci.*, 181(24):5412–5434.

Xiangyang Li, Bo Chen, HuiFeng Guo, Jingjie Li, Chenxu Zhu, Xiang Long, Sujian Li, Yichao Wang, Wei Guo, Longxia Mao, et al. 2022. Inttower: the next generation of two-tower model for pre-ranking system. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 3292–3301.

Xiangyang Li, Bo Chen, Lu Hou, and Ruiming Tang. 2023a. Ctrl: Connect collaborative and language model for ctr prediction. *ACM Transactions on Recommender Systems*.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023b. Towards general text embeddings with multi-stage contrastive learning. *CoRR*, abs/2308.03281.

Jianghao Lin, Xinyi Dai, Yunjia Xi, Weiwen Liu, Bo Chen, Hao Zhang, Yong Liu, Chuhan Wu, Xiangyang Li, Chenxu Zhu, et al. 2025. How can recommender systems benefit from large language models: A survey. *ACM Transactions on Information Systems*, 43(2):1–47.

Huanshuo Liu, Hao Zhang, Zhijiang Guo, Kuicai Dong, Xiangyang Li, Yi Quan Lee, Cong Zhang, and Yong Liu. 2024. Ctrla: Adaptive retrieval-augmented generation via probe-guided control. *arXiv e-prints*, pages arXiv–2405.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

P. Mahalakshmi and N. Sabiyath Fatima. 2022. Summarization of text and image captioning in information retrieval using deep learning techniques. *IEEE Access*, 10:18289–18297.

Yev Meyer, Marjan Emadi, Dhruv Nathawani, Lipika Ramaswamy, Kendrick Boyd, Maarten Van Segbroeck, Matthew Grossman, Piotr Mlocek, and Drew Newberry. 2024. Synthetic-Text-To-SQL: A synthetic dataset for training language models to generate sql queries from natural language prompts.

Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. 2023. MTEB: Massive text embedding benchmark. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2014–2037, Dubrovnik, Croatia. Association for Computational Linguistics.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A human generated machine reading comprehension dataset. In *Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, December 9, 2016*, volume 1773 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. 1994. Okapi at TREC-3. In *Proceedings of The Third Text REtrieval Conference, TREC 1994, Gaithersburg, Maryland, USA, November 2-4, 1994*, volume 500-225 of *NIST Special Publication*, pages 109–126. National Institute of Standards and Technology (NIST).

Ingy Sarhan. 2023. *Open Information Extraction for Knowledge Representation: Triple Extraction and Information Retrieval From Unstructured Text*. Ph.D. thesis, Utrecht University, Netherlands.

Alessandro Del Sole. 2019. Introducing visual studio code. *Visual Studio Code*.

Hongjin Su, Shuyang Jiang, Yuhang Lai, Haoyuan Wu, Boao Shi, Che Liu, Qian Liu, and Tao Yu. 2024. ARKS: active retrieval in knowledge soup for code generation. *CoRR*, abs/2402.12317.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

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

Hangyu Wang, Jianghao Lin, Xiangyang Li, Bo Chen, Chenxu Zhu, Ruiming Tang, Weinan Zhang, and Yong Yu. 2024a. Flip: Fine-grained alignment between id-based models and pretrained language models for ctr prediction. In *Proceedings of the 18th ACM Conference on Recommender Systems*, pages 94–104.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text embeddings by weakly-supervised contrastive pre-training. *CoRR*, abs/2212.03533.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024b. Improving text embeddings with large language models. *CoRR*, abs/2401.00368.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighof. 2023. C-pack: Packaged resources to advance general chinese embedding. *CoRR*, abs/2309.07597.

Weixiang Yan, Yuchen Tian, Yunzhe Li, Qian Chen, and Wen Wang. 2023. CodeTransOcean: A comprehensive multilingual benchmark for code translation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5067–5089, Singapore. Association for Computational Linguistics.

Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023a. RepoCoder: Repository-level code completion through iterative retrieval and generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2471–2484, Singapore. Association for Computational Linguistics.

Wenlin Zhang, Xiangyang Li, Kuicai Dong, Yichao Wang, Pengyue Jia, Xiaopeng Li, Yingyi Zhang, Derong Xu, Zhaocheng Du, Huifeng Guo, et al. 2025. Process vs. outcome reward: Which is better for agentic rag reinforcement learning. *arXiv preprint arXiv:2505.14069*.

Xiangyu Zhang, Yu Zhou, Guang Yang, and Taolue Chen. 2023b. Syntax-aware retrieval augmented code generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 1291–1302, Singapore. Association for Computational Linguistics.

Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhu Chen, and Xiang Yue. 2024. Opencodeinterpreter: Integrating code generation with execution and refinement. *CoRR*, abs/2402.14658.

Radoslaw Z. Ziembinski. 2015. Unsupervised extraction of graph-stream structure for purpose of knowledge retrieval and information fusion. In *Position Papers of the 2015 Federated Conference on Computer Science and Information Systems, FedCSIS 2015, Lódz, Poland, September 13-16, 2015*, volume 6 of *Annals of Computer Science and Information Systems*, pages 53–60.

A Dataset Filtering and Cleaning Process

The dataset used for code retrieval tasks must be of high quality, diverse, and appropriately challenging to ensure the development of robust models. To this end, we implemented a multi-stage filtering and cleaning process that removes irrelevant, incomplete, or overly simplistic code snippets, ensuring the dataset is representative of real-world programming tasks.

A.1 Difficulty

To ensure that each selected dataset presents an appropriate level of difficulty, we have curated a large collection of datasets. For datasets that are overly simplistic and fail to effectively differentiate between model capabilities, we have opted to discard them. The performance of models on these overly simple datasets is summarized in Table 7.

A.2 Deduplication

The first step involved identifying and removing duplicate code snippets. Duplicate examples can distort the model's performance by introducing redundancy, which may lead to overfitting. Using hash-based techniques, we identified exact duplicates. This ensured that each code snippet in the dataset was unique and contributed distinct information.

A.3 Alignment

For code retrieval tasks, the alignment between the code and its associated natural language description is critical. We filtered out examples where comments or documentation were missing, incomplete, or irrelevant to the functionality of the code. This step ensured that the dataset only contained examples with high-quality, informative descriptions that could aid in retrieval tasks.

B Dataset Details

B.1 Reasons for Dataset Selection

- **APPS**: A challenging dataset where state-of-the-art retrieval models achieve an NDCG@10 of only 26.52, highlighting its difficulty.
- **CosQA**: The largest web-based query-to-code dataset, featuring meticulously crafted queries to enhance retrieval performance.
- **Synthetic Text-to-SQL**: The largest synthetic text-to-SQL dataset, enabling robust evaluation of SQL generation models.
- **CodeSearchNet**: A widely used code retrieval dataset covering six programming languages, with a corpus of 1 million documents.
- **CodeSearchNet-CCR**: Specifically designed for the Code Context Retrieval task, which is crucial for improving code completion systems.

Model (NDCG@10)	Human-eval	XcodeEval (sampled 13k)	mbpp
E5-Base-v2	92.41	89.59	95.43
BGE-Base-en-v1.5	93.45	88.47	92.41
GTE-Base-en-v1.5	91.54	87.54	94.45
UniXcoder	92.44	90.85	90.56

Table 7: Model performance comparison on Simple Dataset

- **CodeTransOcean-DL & CodeTransOcean-Contest**: Well-suited for Similar Code Retrieval, encompassing deep learning implementations and competitive programming solutions across multiple languages.
- **StackOverflow QA**: Reflects real-world developer queries, making it ideal for evaluating models that retrieve answers based on complete questions.
- **CodeFeedBack-ST**: A high-quality synthetic QA dataset, beneficial for retrieval-augmented generation and large language models.
- **CodeFeedBack-MT**: Designed for Multi-turn Code QA, presenting a significant challenge due to its extensive dialogue context exceeding 4,000 tokens—far beyond the typical 512-token limit of most models.

Examples of queries and corpora present in each dataset can be viewed in tables 8 and 9.

B.2 Text-to-Code Retrieval Datasets

APPS (Hendrycks et al., 2021). The original APPS dataset is a code generation dataset derived from programming problems shared on open-access sites frequented by programmers, including Codewars, AtCoder, Kattis, and Codeforces, where each example consists of a question description and its corresponding code solution. To adapt the original APPS dataset for retrieval, we use the original problem descriptions as the query to retrieve from a corpus of all code solutions. We retain the original dataset's train-test split and remove examples that do not have a corresponding code solution, resulting in a total of 5,000 samples for the training set and 3,765 samples for the test set.

CoSQA (Huang et al., 2021). The CosQA dataset comprises 20,604 human-annotated labels for pairs of natural language web queries and corresponding code snippets. We retain the original train/dev/test dataset splits of 19,604/500/500 and utilize the natural language web queries as queries to retrieve from a corpus of all code snippets as intended in CoSQA.

Synthetic Text2Sql (Meyer et al., 2024). The Synthetic Text2Sql dataset is a comprehensive collection of high-quality synthetic Text-to-SQL samples, meticulously designed and generated using Gretel Navigator [5]. Each example consists of the following: (1) a problem description which can be resolved using SQL (2) information on the schema of relevant tables used (3) the corresponding SQL code solution (4) meta-data revolving problem described, for instance the type of domain or industry the problem falls under (e.g. healthcare, aerospace etc.) and the nature of the task at hand (e.g. reporting, analytics, dashboarding etc.). Here we use the natural language question description as the query to retrieve from a corpus of corresponding SQL code solutions. We follow the train-test split provided by Gretel where the 105,851 queries are divided into 100,000 queries for train and the remaining 5,851 for test.

B.3 Code-to-Code Retrieval Datasets

CodeSearchNet-CCR (Husain et al., 2019). CodeSearchNet-Code Context Retrieval (CCR) is modified from the original CodeSearchNet dataset with 1 million (docstring, code) pairs sourced from open-source repositories hosted on GitHub. For each example present in the dataset, we randomly divide each code function into two code segments where all the initial segment serves as our query and all latter segments forms the corpus to be retrieved using the corresponding initial segment. The length of each query is uniformly and randomly selected to comprise between 40% and 70% of the total number of characters in the original code or function. We retain the original train/dev/test split of 905k/41k/53k and also retain the partition by their respective language from CodeSearchNet.

CodeTransOcean-DL (Yan et al., 2023). CodeTransOcean-DeepLearning (DL) is derived from the DLTrans, a dataset featured in the Code-

[5] https://docs.gretel.ai/

	Dataset	Query	Relevant-Corpus	Granularity
Text Query to Code Corpus	Apps	You are playing a very popular game called Cubecraft. Initially, you have one stick and want to craft k torches. <*Text omitted for brevity*> For each test case, print the answer: the minimum number of trades you need to craft at least k torches. The answer always exists under the given constraints. <*Example Test Case Omitted*>	`for haaghfj in range(int(input())):` `  x,y,k = list(map(int,input().split()))` `  print(k + (y * k + k - 1 +x-2) // (x - 1))`	Function
	CosQA	python adjacency matrix from edge list	`def get_adjacent_matrix(self):` `  edges = self.edges` `  num_edges = len(edges) + 1` `  adj = np.zeros([num_edges, num_edges])` `  for k in range(num_edges - 1):` `  adj[edges[k].L, edges[k].R] = 1` `  adj[edges[k].R, edges[k].L] = 1` `  return adj`	Function
	Synthetic Text2Sql	Which buildings in the UK have both a green roof and solar panels installed?	`SELECT b.name` `  FROM Building b JOIN GreenRoof gr ON` `  b.id = gr.building_id JOIN` `  SolarPanel sp ON` `  b.id = sp.building_id` `  WHERE b.country = \`UK';`	Entire Code
Code Query to Code Corpus	CodeSearchNet-CCR	`def get_cumulative_spend(key):` `  """` `  Get the sum of spending for this category` `  up to and including the given month.` `  """` `  query = (\`ROUND(SUM(total_ex_vat), 2)` ` AS total ' \`FROM {table} '` `  \`WHERE date <= "{year}-{month:02}-01" '` ` \`AND lot="{lot}" '` `  \`AND customer_sector="{sector}" '` ` \`AND supplier_type="{sme_large}"'.` `  format(` `  table=_RAW_SALES_TABLE,` `  year=key.year,` `  month=key.month,`	`  lot=key.lot,` `  sector=key.sector,` `  sme_large=key.sme_large))` `logging.debug(query)` `result = scraperwiki.sqlite.select(query)` `logging.debug(result)` `value = result[0][\`total']` `return float(result[0][\`total']) if value is not None else 0.0`	Code Snippet
	CodeTransOcean-DL	`import tensorflow as tf` `from d2l import tensorflow as d2l` `net = tf.keras.models.Sequential([tf.` `  keras. layers. Flatten(), tf.keras.` `  layers. Dense(256, activation='relu'),` `  tf.keras.layers.Dense(10)])` `batch_s, lr, num_epochs = 256, 0.1, 10` `loss = tf.keras.losses.` `  SparseCategoricalCrossentropy()` `trainer = tf.keras.optimizers.SGD(lr)` `train_iter, test_iter =` `  d2l.load_data_fashion_mnist(batch_s)` `d2l.train_ch3(net, train_iter, test_iter,` `loss, num_epochs, trainer`	`from d2l import paddle as d2l` `import paddle` `from paddle import nn` `net = nn.Sequential(nn.Flatten(),` `  nn.Linear(784, 256), nn.ReLU(),` `  nn.Linear(256, 10))` `for layer in net:` `  if type(layer) == nn.Linear:` `  weight_attr =` `  paddle.framework.ParamAttr(` `  initializer=paddle.nn.initializer.` `  Normal(mean=0.0, std=0.01))` `  layer.weight_attr = weight_attr` `batch_size, lr, num_epochs = 256, 0.1, 10` `... Code truncated for brevity`	Entire Code
	CodeTransOcean-Contest	`def setup():` `  println(distance("kitten", "sitting"))` `def distance(a, b):` `  costs = []` `  for j in range(len(b) + 1):` `  costs.append(j)` `  for i in range(1, len(a) + 1):` `  costs[0], nw = i, i - 1` `  for j in range(1, len(b) + 1):` `  cj=min(1+min(costs[j],costs[j-1]),` `  nw if a[i-1]==b[j-1] else nw+1)` `  nw, costs[j] = costs[j], cj` `  return costs[len(b)]`	`#include <algorithm>` `#include ...` `template <typename StringType>` `size_t levenshtein_distance(const` `  StringType& s1, const StringType& s2) {` `  const size_t m = s1.size();` `  const size_t n = s2.size();` `  if (m == 0) return n;` `  if (n == 0) return m;` `  std::vector<size_t> costs(n + 1);` `  std::iota(costs.begin(),costs.end(),0);` `  size_t i =...` `... Code truncated for brevity}`	Entire Code

Table 8: Examples of queries and relevant corpora for 6 datasets related to Text-to-Code and Code-to-Code retrieval (i.e. *Apps*, *CosQA*, *Synthetic Text2sql*, *CodeSearchNet-CCR*, *CodeTransOcean-DL*, and *CodeTransOcean-Contest*) in COIR. For brevity, we omit or truncate some portion of the query and corpus.

TransOcean benchmark which focuses on code translation. The original dataset consists of pairs of semantically equivalent deep learning code written using different deep learning libraries namely Tensorflow, Pytorch, PaddlePaddle and MXNet. For each pair of data, we use one of the codes in one of the framework as query to retrieve its counterpart written in a different framework. We follow

	Dataset	Query	Relevant-Corpus	Granularity
Code Query to Code Corpus	CodeSearchNet	`def precedes(self, other):` `  """Returns a boolean indicating whether this element precedes other element"""` `  try:` `  ancestor = next(commonancestors(` `  AbstractElement, self, other))` `  except StopIteration:` `  raise Exception("Elements share` `  no common ancestor")` `  # now we just do a depth first search` `  # and see who comes first` `  def callback(e):` `  .... Code truncated for brevity`	Returns a boolean indicating whether this element precedes the other element	Document String
Hybrid Code Query to Hybrid Code Corpus	StackOverFlow QA	With Rust Regular Expressions, how can I use named capture groups preceding a string? I'm using the sd tool which uses rust regular expressions, and I am trying to use it with a named capture group, however... <*Text ommitted for brevity*> Here is a contrived example to illustrate it: `echo 'abc' | sd -p '(?P<cg>b)' '$cgB'` `# outputs: ac` `# desired output: abBc` `echo 'abc' \| sd -p '(?P<cg>b)' '$cg B'` `# outputs as expected: ab Bc` `# however, places a space there` I've tried `$<cg>B`, `$cg(B)`, `$cg0B`, all don't give `abBc`. I've also checked the rust regex docs however the x flag, and other techniques seem only applicable to the search pattern, not the replace pattern.	We don't need the sd tool to reproduce this behavior. Here it is in pure Rust: `let re = regex::Regex::new(r"(?P<n>b)")` `  .unwrap();` `let before = "abc";` `assert_eq!(re.replace_all(before, "$nB"),` `  "ac");` `assert_eq!(re.replace_all(before, "${n}B"),` `  "abBc");` The brace replacement syntax isn't described in the front documentation but on that of the replace method: The longest possible name is used. e.g., `$1a` looks up the capture group named `1a` and not the capture group at index 1. To exert more precise control over the name, use braces, e.g., `${1}a`. <*Text omitted for brevity*>	Code Snippet mixed with text
	CodeFeedBack-ST	**user:** How would you extend the following code to add the features "undo" and "redo" while maintaining a constant time complexity for adding items? `def add_item(item, list):` `  list.append(item)` `def undo(list):` `  if len(list) > 0:` `  list.pop()` `def redo(undone_list, list):` `  if len(undone_list) > 0:` `  item = undone_list.pop()` `  list.append(item)`	**assistant:** To add the features "undo" and "redo" while maintaining a constant time complexity for adding items, you can use ... Here's an extended version of the code: `def add_item(item, item_list, undone_list,` `  redone_list):` `  item_list.append(item)` `  redone_list.clear() # Clear the` `  # redone_list as new item is added` `def undo(item_list, undone_list,` `  redone_list):` `  if len(item_list) > 0:` `  .... Code omitted for brevity` In this version, we add three new parameters to the functions: 'undone_list'... <*Text omitted for brevity*>	Code Snippet or Functions mixed with text
	CodeFeedBack-MT	**user:** Convert a given 2D point to the polar representation. The point has coordinates (x,y), where x and y are integers between -1000 and 1000, inclusive. Additional Requirement: ...<*Text omitted for brevity*> **assistant:** To convert a 2D point to polar representation, we need to calculate the polar coordinates (r, theta) using the given Cartesian coordinates (x, y). We can use the arctangent function to calculate the value of theta. <*Text omitted for brevity*> Now, let's write the code to convert the given 2D point to polar representation using the atan2 function in Python: `import math` `def convert_to_polar(x, y):` `  r = math.sqrt(x*x + y*y) # calculating the` `  distance from the origin` `  theta = math.atan2(y, x) # calculating the` `  angle in radians using atan2 function` `  return (r, theta)` `... Code truncated for brevity` **user:** <*Text omitted for brevity*>, **assistant:** <*Text omitted for brevity*>, **user:** Please generate the code to meet the original requirements without using arithmetic operations. Clarifying comments and documentation will also be appreciated.	**assistant:** I apologize for any misunderstanding, but converting from Cartesian coordinates to polar coordinates inherently requires mathematical operations – notably, addition, multiplication, and trigonometric operations. These are defined by the mathematical formulas for converting between the two coordinate systems: $$r = sqrt(x^2 + y^2), theta = atan2(y, x)$$ These formulas can't be implemented without using arithmetic operations, due to their inherent mathematical nature. As per your requirement of not using any arithmetic operations to solve this problem, it's essential to understand that it would be impossible to convert to polar coordinates, as the problem mainly depends on these arithmetic operations. Again, I emphasize that this kind of computation relies heavily on arithmetic operations, and it is a significant constraint that would make finding a software or coding solution impossible. However, if there's any related issue or a different problem you need assistance with, I would be more than happy to assist!"	Code Snippet or Functions mixed with text

Table 9: Examples of queries and relevant corpora for 4 datasets related to Code-to-Text and Hybrid Code retrieval (i.e. *CodeSearchNet*, *StackOverflow QA*, *CodeFeedback-ST* and *CodeFeedback-MT*) in COIR. For brevity, we omit or truncate some portion of the query and corpus

the original train/dev/test split of 564/72/180 in CodeTransOcean-DL.

CodeTransOcean-Contest (Yan et al., 2023). Similar to CodeTransOcean-DL, the

CodeTransOcean-Contest dataset is derived from the MultilingualTrans dataset in the CodeTransOcean benchmark. The dataset features a collection of code for various problems, for instance binary tree traversal, sorting algorithms, written in various languages featured in Rosetta Code [6], a programming chrestomathy website. In CodeTransOcean-Contest, we focus on retrieving semantically equivalent C++, Python code pairs as these two languages differs greatly in terms of syntax and language features. We filter the examples in MultilingualTrans that does not contain any C++ nor Python code pairs and we use the Python code in each pair as queries to retrieve its counterpart from all C++ code in the filtered dataset. We retain the dataset split as in MultilingualTrans resulting in a 561/226/221 examples for train/dev/test respectively.

B.4 Code-to-Text Retrieval Datasets

CodeSearchNet (Husain et al., 2019). The CodeSearchNet is a dataset consisting of 1 million (docstring, code) pairs sourced from open-source repositories hosted on GitHub. It contains code and documentation for several programming languages. Instead of retrieving relevant code from the original description, we reverse the roles of the docstring and code by retrieving the relevant docstring using the code as a query. We follow the original CodeSearchNet train/dev/test split of 905k/41k/53k and also retain the partition by their respective language.

B.5 Hybrid Code Retrieval Datasets

StackOverflow QA[7]. We modify the original StackOverflow dataset from Kaggle [8], which contains questions posted by users, the corresponding highest voted answer to the user's questions and tags pertaining to the user's questions. Both the question and the answer typically consist of a mix of text and code snippets. Out of a total of 1,048,576 questions, we randomly sample 19,931 questions and their corresponding answers from the StackOverflow QA dataset. We use the questions raised by the users as a query to retrieve the corresponding answer from other correct answers posted by users. We randomly split the data into train/dev/test sets with a 13,951/3,986/1,994 split respectively.

CodeFeedBack-ST (Zheng et al., 2024). We utilize data from CodeFeedback-Filtered-Instruction to generate retrieval dataset for CodeFeedBack-ST. CodeFeedback-Filtered-Instruction is a collection of instruction-tuning datasets where a language model takes cue from user's instruction in order to generate code to fulfil the user's instructions. The dataset consists of pairs of users instruction and the response to be expected from an assistant where both the user's instructions and the assistant's response could contain a mixture of text and code snippets. We use the user's instructions as query to retrieve the corresponding relevant expected assistant's reply. Of the 156,526 examples present, we split the train and test dataset into 125,221 and 31,307 train and test examples respectively.

CodeFeedBack-MT (Zheng et al., 2024). We utilize examples from Code-Feedback dataset to generate retrieval dataset for CodeFeedBack-Multi Turn (MT). Code-Feedback is a collection of instruction-tuning datasets involving simulated multi-turn dialogues between 2 LLMs; one plays the role of a user and the other plays the role of assistant. To ensure that that the generated replies of the assistant are of high quality, the LLM playing the role of assistant is aided by code compilers alongside multiple generation attempts to arrive at a desirable response. Each example comprises a sequence of exchanges alternating between the user and assistant, starting with the user and ending with the assistant. To construct CodeFeedBack-MT, we split each sequence into 2 portions, the first portion consists of the initial dialogue history, prior to the last reply by the assistant, and second part is made up of the final reply by the assistant. We use the initial portion of the dialogue as query to retrieve the corresponding reply by the assistant. We split the dataset into 53,106 and 13,277 for train and test respectively.

C Metric

C.1 NDCG Metric

Normalized Discounted Cumulative Gain (NDCG) is a popular metric used to evaluate the quality of rankings, particularly in information retrieval and recommender systems. It measures the usefulness, or *gain*, of an relevant item based on its position in the result list, discounted logarithmically by the position.

The Discounted Cumulative Gain (DCG) is the sum of the gains of relevant items, discounted log-

[6] https://rosettacode.org/wiki/Rosetta_Code

[7] https://www.kaggle.com/datasets/stackoverflow/stacksample/data

[8] https://www.kaggle.com/

arithmically by their positions in the ranking. The normalized version, NDCG, compares the DCG of the ranked list to the DCG of the ideal ranking. This normalization ensures that the score is within the range [0, 1].

DCG Calculation:

The DCG at position p is calculated as:

$$\text{DCG}_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

where rel_i is the relevance score of the item at position i.

IDCG (Ideal DCG) Calculation:

The Ideal DCG (IDCG) is the DCG of the ideal ranking. This is calculated by sorting all items by their relevance scores in descending order and then computing the DCG using the same formula.

$$\text{IDCG}_p = \sum_{i=1}^{p} \frac{2^{rel_i^*} - 1}{\log_2(i+1)}$$

where rel_i^* is the relevance score of the item at position i in the ideal ranking.

NDCG Calculation:

The NDCG at position p is the ratio of the DCG at position p to the IDCG at position p:

$$\text{NDCG}_p = \frac{\text{DCG}_p}{\text{IDCG}_p}$$

This normalization ensures that NDCG values are bounded between 0 and 1, making comparisons across different queries or datasets meaningful.

By evaluating the NDCG, we can get a sense of how well our ranking system is performing relative to the best possible ranking. This is particularly useful when the relevance varies across items and lower-ranked items are less likely to be seen and thus should contribute less to the overall evaluation metric.

C.2 Weighted Jaccard Similarity

The weighted Jaccard similarity $J(S,T)$ (Ioffe, 2010) is calculated as the unique word overlap for all words present in both the datasets. We define the normalized frequency for an unique word k in a dataset to be the frequency of word k divided over the sum of frequencies of all words in the dataset.

Let S_k be the normalized frequency of word k in the source dataset S and T_k for the target dataset T respectively. The weighted Jaccard similarity between S and T is defined as:

$$J(S,T) = \frac{\sum_k \min(S_k, T_k)}{\sum_k \max(S_k, T_k)}$$

where the sum is over all unique words k present in datasets S and T.

D Implementation Details

D.1 Implementation Environments

The experimental setup for performance and efficiency evaluation was conducted utilizing a Tesla V100 GPU with 32GB of memory and Intel Titan CPU and CUDA 11.2. The versions of the software utilized in this study are PyTorch 2.0.1 and Transformers 4.38.1. We utilized the Faiss IndexFlat(Johnson et al., 2021) to store vector embedding index.

D.2 Retriever Resources

The huggingface models or API used of retrieval model used in CoIR is listed in Table 10.

D.3 Dataset Resources

The urls of dataset resources used in CoIR is listed in Table 11.

E Dataset Explanation of Reality

It is important to note that the current settings align with prevalent code retrieval benchmark methodologies in the field, such as CodeSearchNet and CoSQA, where the retrieved documents are considered the final answers to queries. Additionally, we have considered scenarios where retrieval serves as a stepping stone rather than just an endpoint. Specifically, our code-to-text and code-to-code tasks are designed to evaluate the model's ability to retrieve relevant text and code snippets that supplement and enhance the final answer.

Furthermore, we have developed a new dataset from Stack Overflow. This dataset is carefully curated and filtered from the original Stack Overflow dump. In this dataset, the provided answers are not always the final solution to the queries. Instead, they may represent a strategy for solving the problem or serve as a component of the solution, thus reflecting the nuanced and iterative nature of real-world code generation and retrieval tasks.

F Clarification of Innovation

We are not merely integrating existing datasets; rather, we are introducing new datasets and tasks. CodeSearchNet-CCR and StackOverflowQA are

Model	Public Model Checkpoints (Link)
Contriever	https://huggingface.co/facebook/contriever-msmarco
E5-base	https://huggingface.co/intfloat/e5-base-v2
BGE-Base	https://huggingface.co/BAAI/bge-base-en-v1.5
GTE-Base	https://huggingface.co/Alibaba-NLP/gte-base-en-v1.5
UniXcoder	https://huggingface.co/microsoft/unixcoder-base
BGE-M3	https://huggingface.co/BAAI/bge-m3
E5-Mistral	https://huggingface.co/intfloat/e5-mistral-7b-instruct
OpenAI-Ada-002	https://openai.com/
Voyage-Code-002	https://www.voyageai.com/

Table 10: Publicly available model links used for evaluation in **CoIR**.

Corpus	Website (Link)
APPS	https://huggingface.co/datasets/codeparrot/apps
CoSQA	https://github.com/microsoft/CodeXGLUE/tree/main/Text-Code/NL-code-search-WebQuery
Synthetic Text2sql	https://huggingface.co/datasets/gretelai/synthetic_text_to_sql
CodeSearchNet	https://huggingface.co/datasets/code-search-net/code_search_net
CodeTransOcean	https://huggingface.co/datasets/WeixiangYan/CodeTransOcean
Code-FeedBack	https://huggingface.co/datasets/m-a-p/Code-Feedback
CodeFeedBack-Filtered-Instruction	https://huggingface.co/datasets/m-a-p/CodeFeedback-Filtered-Instruction

Table 11: Corpus Name and Link used for datasets in **CoIR**.

datasets we created ourselves. StackOverflowQA was extracted from the original StackOverflow dump, and we introduced this dataset to ensure that the simulated tasks closely mirror real-world scenarios, such as searching for answers when encountering code issues.

Additionally, we introduced a new task, Code Context Retrieval, which involves retrieving the most relevant code segment that completes a given initial segment of code. This task is critical for code completion purposes, and the CodeSearchNet-CCR dataset was specifically created for this task.

Previously, only CosQA and CodeSearchNet were straightforwardly usable as code retrieval benchmarks. We have supplemented these with eight additional datasets, which we meticulously cleaned, manually filtered, and standardized. This process was quite labor-intensive.