CodeXEmbed: A Generalist Embedding Model Family for Multiligual and Multi-task Code Retrieval

Anonymous ACL submission

Abstract

Despite the success of text retrieval in many NLP tasks, code retrieval remains a largely underexplored area. Most text retrieval systems are tailored for natural language queries, often neglecting the specific challenges of retrieving code. This gap leaves existing models unable to effectively capture the diversity of programming languages and tasks across different domains, highlighting the need for more focused research in code retrieval. To address this, we present CODEXEMBED, a family of large-scale code embedding models ranging from 400M to 7B parameters. Our novel training pipeline unifies multiple programming languages and transforms various code-related tasks into a common retrieval framework, enhancing model generalizability and retrieval performance. The 7B model sets a new state-of-the-art (SOTA) in code retrieval, outperforming the previous leading model, Voyage-Code, by over 20% on CoIR benchmark. In addition to excelling in code retrieval, our models demonstrate competitive performance in text retrieval BeIR benchmark, offering versatility across domains. Experimental results demonstrate that improving retrieval performance significantly enhances end-to-end Retrieval-Augmented Generation (RAG) performance for code-related tasks.

1 Introduction

Large Language Models (LLMs) have demonstrated exceptional performance across numerous Natural Language Processing (NLP) tasks. However, they often struggle to produce faithful answers and may lack up-to-date or domain-specific knowledge. To bridge this gap, retrieval-augmented generation (RAG) (Cai et al., 2022; Cheng et al., 2024) techniques have gained prominence, integrating Information Retrieval (IR) systems with LLMs to enhance their access to relevant external information. This synergy has drawn significant attention recently, leading to the development of various retrieval models (Wang et al., 2022; Chen et al., 2024) based on BERT (Kenton and Toutanova, 2019) and other LLMs with sizes exceeding 1 billion parameters (Wang et al., 2023; Moreira et al., 2024; Meng et al., 2024). Despite these advancements, standard IR methods, while effective in text-based retrieval, often fall short in specialized domains such as code retrieval (Husain et al., 2019).

Code retrieval is critical for accelerating development processes and improving code quality. Unlike general text retrieval, code retrieval enables developers to quickly locate relevant code snippets, explanations, bug analyses, summaries, and similar instances. Effective code retrieval systems are now integrated into commercial products like VS Code (Del Sole, 2021) and GitHub Copilot (Wermelinger, 2023; Yetiştiren et al., 2023), enhancing productivity. Code-RAG systems (Parvez et al., 2021; Liu et al.; Jimenez et al., 2024; Wang et al., 2024) also leverage code retrieval to minimize hallucinations in generated code from LLMs, ensuring more accurate outputs.

However, traditional text retrieval models often struggle with code because they focus on linguistic patterns, while code retrieval must handle elements like syntax, variable dependencies, control flow, and API usage. Despite the importance of code-specific models, existing ones like CodeBERT (Feng et al., 2020), CodeGPT (Lu et al.), and UniXcoder (Guo et al., 2022) are based on smaller BERT models (Kenton and Toutanova, 2019). While large-scale models for retrieval have become popular, only Voyage-Code has followed this approach for code retrieval, but it remains a closed model, leaving a gap for open-source, large-scale code retrieval models.

In this work, we introduce CODEXEMBED, a family of open-source embedding models tailored for both code and text, available in sizes of 400 million, 2 billion, and 7 billion parameters. CODEXEMBED introduces a generalizable training framework that converts multiple programming lan-

guages and diverse code-related tasks into retrieval tasks. Our approach handles 12 programming languages and five distinct code retrieval categories across eight different code tasks, including code-to-text, text-to-code, code-to-code, text-to-text and hybrid text and code tasks. This comprehensive setup enables CODEXEMBED to generalize effectively across various code domains. Our contributions can be summarized as follows:

- We propose a generalizable training approach for code embedding that transforms diverse programming languages and code-related tasks into retrieval tasks. This method allows for a unified framework that effectively handles multiple languages and task types, improving retrieval performance across a wide range of code data.

- Our 7B model, the open-source code embedding model to date, achieves state-of-the-art performance on code retrieval CoIR benchmark, setting a new standard for code retrieval. We also evaluate CODEXEMBED on RepoEval and SWE-Bench-Lite, demonstrating that improved retrieval significantly enhances end-to-end Retrieval-Augmented Generation (RAG) performance for code tasks.

- In addition to the 7B model, we provide smaller models (400M and 2B) that still outperform the previous state-of-the-art Voyage model. Furthermore, our models demonstrate competitive performance on text retrieval BEIR benchmark, making them versatile across both code and text retrieval tasks.

2 Method

We transform general code-related tasks into a unified retrieval framework by categorizing them into distinct retrieval settings. As shown in Figure 1, each setting serves a specific purpose in mapping between code and text, enhancing the model's ability to generalize across various retrieval tasks.

Text-to-Code Retrieval. The Text-to-Code retrieval setting focuses on retrieving relevant code documents based on a given textual query. This setup is particularly useful in scenarios where natural language descriptions are mapped to code solutions. Several code generation tasks can be transformed to fit into this retrieval paradigm.

One such task is *code contest generation* (Billah et al., 2024; Kadir et al., 2024), where a natural language description of a programming problem is provided, and the goal is to retrieve or generate a code snippet that correctly solves the problem. This setting closely aligns with the Text-to-Code retrieval framework as it involves finding code solutions based on a question or problem statement. Another example is the *Text-to-SQL* task (Finegan-Dollak et al., 2018; Li et al., 2024a), where the aim is to generate or retrieve a SQL query based on a user's natural language question. The task fits well within the Text-to-Code category, as it requires understanding the intent of the query and retrieving or generating the appropriate SQL code that matches the given description.

Code-to-Text Retrieval. The Code-to-Text retrieval setting is designed to retrieve relevant textual documents or descriptions based on a given code query. This setup is particularly valuable in scenarios where developers or systems seek to understand or explain code by mapping it to human-readable documentation or summaries.

One such task is *code summarization* (Sontakke et al., 2022; Sun et al., 2024), where a code file, or even an entire repository, is provided as input, and the goal is to generate a concise textual summary of the code's functionality. This task aligns with the Code-to-Text retrieval setting, as it involves taking potentially complex code and producing a human-readable description or explanation, which is critical for documentation, knowledge sharing, and onboarding in software projects.

Code-to-Code Retrieval. The Code-to-Code retrieval setting focuses on retrieving relevant code documents based on a code query. This retrieval setting is particularly useful for tasks that involve understanding, transforming, or complementing code snippets. Several tasks naturally align with this framework, making it an essential part of the broader code retrieval ecosystem.

One such task is *code translation* (Pan et al., 2024; Karanjai et al., 2024), where the goal is to translate code snippets from one programming language to another. For instance, a developer might provide a code snippet in Python and seek to translate it into C++ or Java. This task fits well into the Code-to-Code retrieval setting, as it involves finding the equivalent functionality in a different programming language, ensuring the translated code maintains the same logic and structure.

Another important task is *code completion* (Ding et al., 2024; Phan et al., 2024; Liu et al., 2024), where a partially written code snippet is provided,

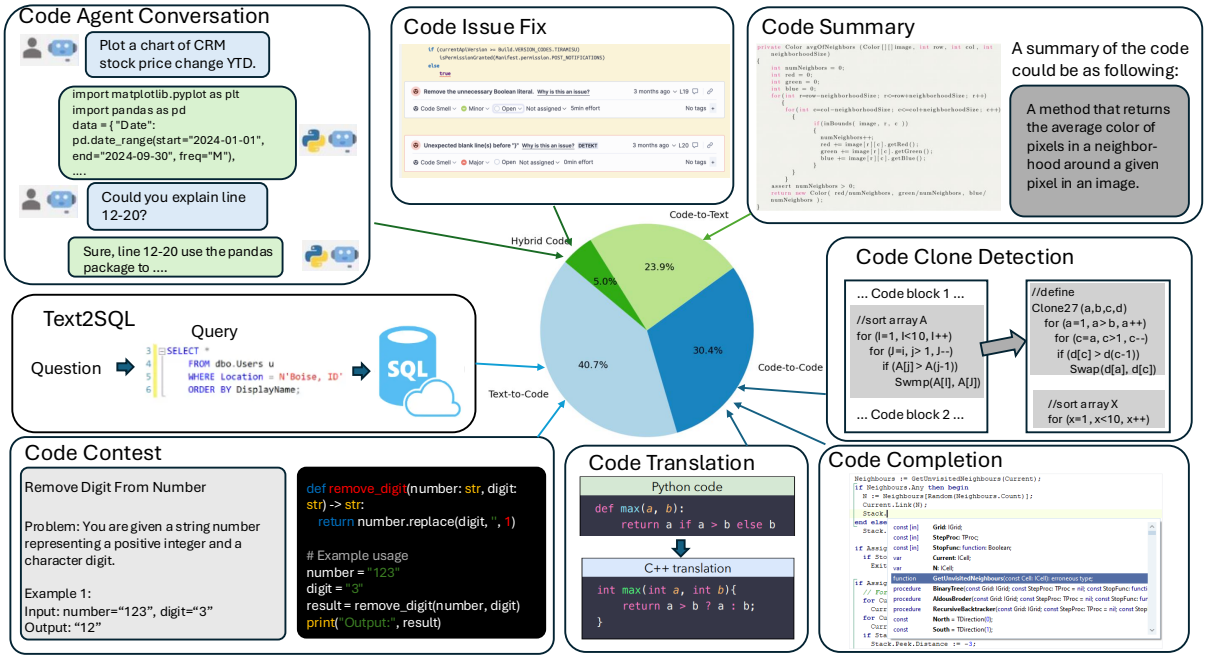

Figure 1: The code training data of CODEXEMBED contains four parts: Text-to-Code, Code-to-Code, Code-to-Text and Hybrid Code. Each Categories contains several types of code tasks.

and the system needs to retrieve or generate the missing portions to complete the function or module. This is commonly seen in integrated development environments (IDEs) where autocomplete features assist developers by suggesting the next logical lines of code based on the current context. Code completion aligns perfectly with the Code-to-Code retrieval setting, as it requires matching the partial input code with relevant snippets that seamlessly complement the existing code.

Additionally, *code clone detection* (Martinez-Gil, 2024), which involves identifying code snippets that are functionally or structurally identical but may differ in syntax or formatting, also fits within this retrieval framework. Clone detection is crucial for identifying redundant code, refactoring codebases, and ensuring consistency across different sections of a project. This task leverages the Code-to-Code retrieval setting by retrieving code snippets that exhibit similar or identical behavior, even if they appear different at the surface level.

Hybrid Code Retrieval. The hybrid code retrieval setting involves retrieving a combination of both code and textual documents in response to a hybrid query, which may contain both code and natural language. This setting is particularly useful for tasks that require understanding and integrating both code and textual context to generate meaningful responses or retrieve relevant documents.

One such task is *code agent conversation* (Arteaga Garcia et al., 2024; Jin et al., 2024), where a code agent must generate either a code snippet or a textual response based on a user's question. This task requires the agent to interpret the user's question, which might contain both code and natural language, and retrieve or generate the appropriate code solution or explanation. The hybrid query allows the agent to navigate between technical (code) and descriptive (textual) domains to provide a comprehensive response.

Another key task is *code issue fixing* (Yang et al., 2024; Jimenez et al., 2024), where a hybrid query containing a description of the code issue and the problematic code snippet is provided. The goal is to generate code that resolves the issue. By incorporating both the textual description (e.g., error message) and the code itself, hybrid retrieval allows the model to identify relevant code snippets, detect similar issues, and propose fixes that align with the overall project. This makes hybrid retrieval essential in complex development environments, where both debugging and problem-solving require a nuanced understanding of both code and text.

Text-to-Text Retrieval. In order to capture the model's general retrieval ability across both code and natural language domains, we incorporate text-to-text retrieval data into the training pipeline. This allows the model to perform well in traditional text

retrieval tasks, enhancing its versatility. The experiments study the ratio of text and code data to determine the optimal balance between the two, aiming to ensure the model's effectiveness in both text and code retrieval tasks without sacrificing performance in either domain.

Retrieval Training. To train a unified code and text retrieval model, we transform all the aforementioned tasks into a query-and-answer format, where either the query or the answer can be a text or code document. This unified format allows us to handle a variety of tasks such as Text-to-Code, Code-to-Text, and Code-to-Code retrieval within the same training framework. The training loss is designed as a contrastive loss (Liu et al., 2021), which aims to maximize the similarity between the query and the correct answer while minimizing the similarity to the negative samples. The loss function is represented as:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\left[\log\frac{\exp(q_i \cdot d_i^+)}{\exp(q_i \cdot d_i^+) + \sum_{j=1}^{K}\exp(q_i \cdot d_j^-)}\right] \tag{1}$$

where q_i represents the query, which can be either text or code. The term d_i^+ refers to the corresponding positive document, which could also be in the form of text or code. On the other hand, d_j^- denotes a set of hard negative documents, which are selected to be similar in context to the query but incorrect, making the retrieval task more challenging. The similarity between the query and a document is measured by $\text{sim}(q, d)$. The variable N represents the total number of training samples, indicating the number of queries used during training, while K defines the number of hard negatives associated with each query.

3 Experiments

Datasets. COIR (Li et al., 2024b) is a comprehensive benchmark specifically designed for code retrieval tasks. COIR covers a wide range of retrieval challenges, including 8 fine-grained retrieval subtasks, spanning 14 major programming languages. The dataset is composed of 10 distinct datasets, with a total corpus exceeding 2 million entries. **BEIR** (Thakur et al.) is a widely-adopted benchmark designed for text retrieval tasks. BEIR encompasses a diverse set of retrieval challenges, covering 9 distinct tasks across various domains such as question answering, duplicate detection, fact-checking, and more. It supports retrieval over a wide range of datasets and provides a standardized benchmark for evaluating text retrieval models across different domains.

Implementation Details. We conduct general training on our proposed code and text pair dataset using three model sizes: 400M, 2B, and 7B. For the $\text{CodeXEmbed}_{\text{400M}}$, we use the base model *Alibaba-NLP/gte-large-en-v1.5* (Li et al., 2023b), applying full model fine-tuning. For the $\text{CodeXEmbed}_{\text{2B}}$, we initialize our embedding model from the generation model *google/gemma-2-2b-it* (Team et al., 2024), using low-rank adaption(LoRA) (Hu et al., 2022) with a rank of 8. For the $\text{CodeXEmbed}_{\text{7B}}$, we initialize our embedding model from the generation model *mistralai/Mistral-7B-Instruct-v0.3*, also using LoRA with a rank of 8. Following prior work (Meng et al., 2024), we apply: (i) last token pooling for $\text{CodeXEmbed}_{\text{2B}}$ and $\text{CodeXEmbed}_{\text{7B}}$, and (ii) beginning token pooling for $\text{CodeXEmbed}_{\text{400M}}$ to generate semantic vector representations (Li et al., 2023b). Cosine similarity is used to compute the similarity between query and corpus for ranking. The batch size is set to 1024 across all three model sizes, with 7 hard negatives. The learning rate is set to $5e^{-5}$, and the end learning rate to $5e^{-6}$, with linear decay and a 50-step warmup. To improve training efficiency and reduce GPU memory usage, we adopt gradient caching (Gao et al., 2021). The more implementation details can be found in Appendix A.2.

Baseline Models. For code-domain-specific models, we included UniXcoder (Guo et al., 2022) and Voyage-Code-002[1], both of which are pre-trained on code data, making them strong baselines for comparison. For general retrieval models, we evaluated E5-Base (Wang et al., 2022), GTE-Base (Li et al., 2023b), BGE-Base (Xiao et al., 2023), Contriever (Izacard et al., 2023), E5-Mistral (Wang et al., 2023), BGE-M3 (Chen et al., 2024), NV-Embed-V2 (Moreira et al., 2024), SFR-V2 (Meng* et al., 2024) and OpenAI-Ada-002[2].

Evaluation Metrics. In code retrieval, selecting the right evaluation metric is key for assessing both ranking sensitivity and relevance. Building on prior work (Wang et al., 2013), Normalized Discounted Cumulative Gain (NDCG) is preferred for its ability to account for both rank order and varying relevance, unlike precision, recall or MRR. Therefore, we use NDCG@10 to evaluate performance on CoIR and BEIR.

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

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

Model	Text-to-Code			Code-to-Text	Code-to-Code			Hybrid Code			Avg
	Apps	CosQA	Text2SQL	CSN	CSN-CCR	CodeTrans -Contest	CodeTrans -DL	StackOverFlow QA	CodeFeedBack -ST	CodeFeedBack -MT	
Baselines											
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	55.19	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	51.04	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	67.68	65.35	28.74	**56.26**
General Training											
CodeXEmbed$_{400M}$	48.57	34.05	58.96	72.53	80.15	75.67	**34.85**	89.51	78.87	45.75	61.89
CodeXEmbed$_{2B}$	74.99	**36.31**	59.00	73.50	85.77	86.63	33.17	90.54	**81.15**	53.08	67.41
CodeXEmbed$_{7B}$	**85.22**	33.27	64.57	**78.84**	**86.77**	**90.64**	32.31	**94.25**	80.93	**57.83**	**70.46**
In-domain Training											
CodeXEmbed$_{400M}$	45.91	41.28	61.29	81.23	93.74	82.72	**40.81**	92.35	83.36	61.51	68.42
CodeXEmbed$_{2B}$	76.86	40.47	78.42	87.87	97.66	90.30	38.57	94.47	86.36	65.51	75.65
CodeXEmbed$_{7B}$	**85.38**	**42.47**	**78.94**	**89.67**	**97.95**	**94.45**	40.46	**96.33**	**87.53**	**68.83**	**78.20**

Table 1: Performance of the Pretrained CODEXEMBED Model Family across various tasks, including Text-to-Code, Code-to-Text, Code-to-Code, and Hybrid Code, along with their average scores. CSN stands for CodeSearchNet.

3.1 General Training Evaluation

In the *General Training* block of Table 1, we present the results of models trained exclusively on our proposed general training data, without using any CoIR in-domain data. When averaged over all 10 datasets in the CoIR benchmark, our large model (7B) achieves the best results, exceeding the SOTA code-domain specifc model Voyage-Code-002 by over 20%, which shows our general code and text training stage significantly improves model performance on code tasks.

The performance gap between our 2B and 7B models is 3.05, a reasonable difference that allows users to choose based on the tradeoff between performance and cost. Even our small model (400M) outperforms the best comparable model, Voyage-Code-002, and greatly surpasses the similar-sized BGE-M3, which scores 39.31, while our model achieves 61.89. On CoIR CodeSearchNet, Voyage-Code-002 superior performance may be due to using CodeSearchNet as part of their training data, unlike our general training approach.

3.2 In-domain Training Evaluation

We further trained the model on the CoIR in-domain dataset. As shown in the *In-domain Training* block of Table 1, further training on in-domain data results in consistent performance improvements across all model sizes. Specifically, CodeXEmbed$_{400M}$ improves by 6.5 points, CodeXEmbed$_{2B}$ by 8.24 points, and CodeXEmbed$_{7B}$ by 7.74 points on average across all 10 datasets. In Figure 2, the top of each bar represents the improvement from in-domain training. Among all categories, the Code-to-Text category shows the largest improvement, even outperforming Voyage-Code-002. In other categories, the model also achieves over a 5-point improvement.

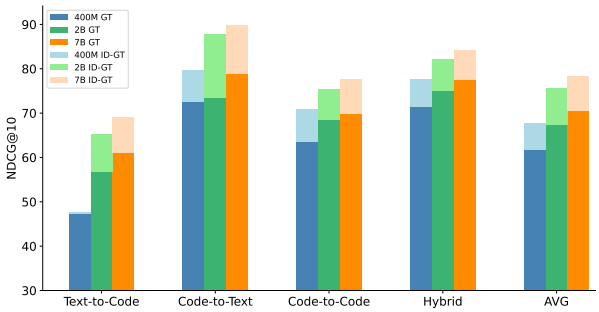

Figure 2: The performance comparison between General Training (GT) and In-domain Training (ID) across three model sizes (400M, 2B, and 7B) on different CoIR categories and the overall average.

3.3 Unified Text and Code Retrieval

To evaluate the text and code retrieval capabilities within a single embedding model, we also present the BEIR text retrieval performance of CODEXEMBED across various sizes. As shown in Table 2, our 7B model achieves an average score of over 60 across 15 datasets, placing it among the top tier on the MTEB leaderboard[3]. Compared to E5-Mistral-7B-Instruct (E5-Mistral) (Jiang et al., 2023), which

[3] https://huggingface.co/spaces/mteb/leaderboard

Model	BM25	gte-large 400M	gte-Qwen2 1.5B	E5-Mistral 7B	CODEXEMBED 400M	CODEXEMBED 2B	CODEXEMBED 7B
MS MARCO	22.8	42.93	**43.36**	43.06	42.77	41.26	42.05
TREC-Covid	65.6	77.49	85.38	**87.03**	77.47	84.58	79.04
NFCorpus	32.5	36.95	39.34	38.58	35.76	41.56	**43.14**
NQ	32.9	56.08	56.08	63.53	63.38	67.25	**74.11**
HotpotQA	60.3	68.18	64.00	75.72	74.93	74.39	**79.33**
FiQA	23.6	**63.23**	63.23	56.81	60.20	56.17	60.41
ArguAna	31.5	**72.11**	54.70	61.65	69.67	61.39	63.58
Touche-2020	**36.7**	22.55	27.89	26.27	20.18	26.10	25.80
CQADupStack	29.9	42.16	44.76	42.97	46.07	47.46	**51.45**
Quora	78.9	**89.67**	89.64	89.61	89.05	89.27	89.51
DBPedia	31.3	46.30	48.69	48.89	46.68	47.33	**49.27**
Scidocs	15.8	**26.35**	24.98	16.32	25.05	23.36	25.25
Fever	75.3	93.81	91.57	87.84	**93.86**	89.03	91.94
Climate-Fever	21.3	**48.36**	42.91	38.35	42.70	32.08	36.93
Scifact	66.5	82.43	78.44	76.42	87.37	84.79	**89.10**
Average	41.7	57.91	58.29	56.87	58.34	57.73	**60.06**
Best on	1	5	1	1	1	0	6

Table 2: Comparison of performance across text retrieval BEIR datasets with different model size.

is trained on both text and synthetic data, initialized from the Mistral series, our model employs single-stage training with both code and text data. Our model in a performance improvement of 3.19 points over E5-Mistral.

In the 400M models, CODEXEMBED achieves a 0.43 performance boost compared to GTE-large (Li et al., 2023b), on which it is trained. This underscores the advantage of our approach, demonstrating the potential to improve text retrieval performance by incorporating code data. There are not many 2B-sized language models available, and we selected Gemma 2B (Team et al., 2024) due to its strong performance in code retrieval. For text retrieval, it achieves comparable performance to gte-Qwen2 of similar size.

Model	NV-Embed-V2 7B	SFR-v2 7B	CODEXEMBED 7B
CoIR	59.10	61.48	**70.46**
BEIR	**62.65**	60.18	60.06
AVG	60.88	60.83	**65.26**

Table 3: Comparison of code and text retrieval benchmarks between our model and the top models on the MTEB leaderboard.

We present the performance of leading text retrieval models from the MTEB leaderboard[4] on both code and text tasks, in Table 3. Compared to the top-ranked model, NV-Embed-V2 (Moreira et al., 2024)[5], our model outperforms it by 4.38 points, with an average score across text and code retrieval datasets.

[4]https://huggingface.co/spaces/mteb/leaderboard

[5]Top-performing model as of 10/15/2024.

3.4 Retrieval-Augmented Code Generation

In this section, we explore how different retrievers influence the final code completion and issue resolution performance in repository-level tasks.

3.4.1 RepoEval

To address this, we utilize RepoEval (Zhang et al., 2023) for repository-level code completion. While RepoEval consists of three splits (function, class, and module), we report results only on the function split, as it is the only one that supports execution-based evaluation. We adopt Pass@1 as our evaluation metric, which measures the accuracy of the top-1 generated code passing the provided test cases.

For code generation, we supply the top-5 retrieved code snippets as input to the GPT-3.5-turbo generation model. As shown in Table 4, all sizes of CODEXEMBED outperform the canonical setup. While some files may not contain direct solutions, as in the canonical documents, they often include valuable function definitions or usage examples that improve code generation outcomes. This suggests that our embeddings effectively capture the repository structure and retrieve contexts that implicitly support problem-solving.

3.4.2 SWE-Bench-Lite

In our experiments, we use SWE-bench-Lite[6], a curated subset of 300 problems from the original SWE-bench benchmark. It focuses on resolving GitHub issues by requiring models to modify multiple files to pass test cases, providing a manageable and reproducible dataset. SWE-bench-Lite also in-

[6]https://huggingface.co/datasets/princeton-nlp/SWE-bench_Lite

Model	Repo-Level	
	RepoEval	SWE-Bench-Lite
None	23.9	0.7
BM25	30.8	1.0
Voyage, code	43.2	0.7
OpenAI, small	48.0	0.3
OpenAI, rerank	49.6	0.0
CodeXEmbed_{400M}	52.5	0.6
CodeXEmbed_{2B}	**66.3**	1.6
CodeXEmbed_{7B}	57.8	**2.6**
Gold	39.1	2.7

Table 4: Performance of repository-level code retrieval augmented generation using gpt-3.5-turbo.

Model	gte-Qwen2		Gemma-v2	SFR-v2		Mistral
	Initial	GT	GT	Initial	GT	GT
Size	1.5B		2B	7B		7B
CoIR	62.96	68.52	67.41	61.28	69.72	**70.40**
BEIR	58.29	59.12	57.73	60.18	**60.62**	60.06

Table 5: Comparison of base models: Retrieval vs. Generation Models. GT represents our general training.

Model	Code-Specific LLMs		General-Domain LLMs	
	StarCoder-v2 3B	DeepSeek-Coder 6.7B	Gemma-v2 2B	Mistral 7B
CoIR	66.95	**71.66**	67.41	70.40
BEIR	49.06	50.22	57.73	**60.06**

Table 6: Comparison of base models: Code-specific vs. General-domain Generation Models.

cludes a pre-configured Docker container, ensuring consistent evaluation across systems and further standardizing the testing environment. Our results show that incorporating improved retrieval methods can significantly enhance the end-to-end performance of code retrieval-augmented generation, bringing it closer to using gold content and boosting problem-solving efficiency and accuracy.

3.5 Impact of the Base Models

To understand the base model's impact, we examine: (1) if training from a text retrieval model outperforms a generation model, and (2) if a code-specific generation model offers more advantages than a general language model.

3.5.1 Embedding Models v.s. LLMs

As shown in Table 5, the text retrieval model offers a stronger starting point, and additional training with our approach enhances both its text and code retrieval capabilities. For instance, gte-Qwen2 [7]'s CoIR performance improves from 62.96 to 68.52, while its text performance increases from 58.29 to 59.12. In contrast, the text generation model requires more extensive fine-tuning to reach similar performance. However, the advantage of text retrieval models can sometimes hinder code retrieval performance, as seen with SFR-V2 (Meng* et al., 2024) underperforming compared to Mistral in specific tasks. This suggests that prior knowledge from text-focused models may not always transfer well to code-specific scenarios. To pursue a more general training approach, we chose to train using a generation model rather than a text retrieval model.

[7] https://www.aimodels.fyi/models/huggingFace/gte-qwen2-15b-instruct-alibaba-nlp

3.5.2 Code-Specific LLMs v.s. General LLMs

We evaluate whether to choose code-specific models (Lozhkov et al., 2024; Guo et al., 2024) or general LLMs (Jiang et al., 2023; Team et al., 2024). As shown in Table 6, code-specific LLMs excel in code tasks but underperform in text tasks, while general LLMs perform well in both. This suggests that recent advancements in general LLMs have integrated code data into their training (Team et al., 2024), and this capability can be effectively transferred to code retrieval. This finding highlights the versatility of general LLMs, making them viable for both text and code retrieval without the need for specialized models.

3.6 Programming Language Transferability

We aim to explore the diversity of programming languages and their unique features. The details of our code training dataset, including language coverage, are provided in Appendix A.1. Our dataset comprises 12 programming languages, with Python representing the highest percentage of the data. For testing, we selected Python and Java due to their distinct programming paradigms: Python is known for its scripting capabilities and Java for its strong object-oriented design. This selection allows us to evaluate our model's performance across a range of programming styles, reflecting the versatility and adaptability of the embedding model.

As shown in Table 7, training with all 12 programming languages yields the best average performance across 7 target languages, compared to training with a single language. However, training on Java-only consistently achieves the highest performance for Java and delivers comparable re-

	Model Size	Python	Java	Go	PHP	Javascript	Ruby	SQL	AVG	Δ-All
All	400M	59.70	72.09	79.28	70.96	70.18	73.01	58.87	69.16	-
	2B	65.73	79.40	81.01	79.39	76.76	78.42	58.23	74.14	-
	7B	**67.56**	81.36	**85.76**	**81.89**	78.37	**83.11**	**65.74**	**77.68**	-
Python	400M	55.43	49.57	49.56	40.57	44.57	44.57	44.21	46.93	−32.1%
	2B	60.86	71.46	68.69	57.88	62.95	72.21	56.09	64.31	−13.3%
	7B	67.02	78.35	81.56	70.88	75.73	79.19	62.56	73.61	−5.2%
Java	400M	54.79	78.37	72.94	70.85	69.85	68.78	50.31	66.56	−3.8%
	2B	63.03	82.17	79.35	80.30	76.00	76.76	58.03	73.66	−0.6%
	7B	66.24	**84.25**	83.72	80.18	**79.82**	82.82	64.59	77.37	−0.4%

Table 7: Python/Java indicates that only the Python and Java portions were used to train CODEXEMBED, while All indicates that all programming languages were used to train CODEXEMBED. Δ-All represents the difference between the Python/Java AVG score and the All AVG score for the same model size.

sults to using all languages across all model sizes. For example, the Java-only 7B model scores 77.37, while the all-languages model scores 77.68. When comparing models trained exclusively on Python or Java, the Java-trained model consistently outperforms. This may be because modern language models are already heavily trained on Python during the generation phase, so relying solely on Python in the retrieval phase may miss important nuances, resulting in suboptimal performance.

4 Related Work

4.1 Retrieval Models for Text and Code

Text retrieval models have significantly advanced by exploiting supervision from natural language inference tasks and labeled query-document pairs, such as the MS-MARCO passage ranking dataset (Bajaj et al., 2016), to train effective text embeddings (Izacard et al., 2021; Wang et al., 2022; Xiao et al., 2024). Recently, researchers have leveraged large language models (LLMs) (Jiang et al., 2023) as the base for training retrieval models, resulting in state-of-the-art performance. Notable examples include E5-Mistral (Wang et al., 2023), SFR-Embedding (Meng et al., 2024), and NV-Embedding (Lee et al., 2024). While numerous models have been developed for text retrieval tasks, few have focused specifically on code retrieval. Among the few are the Voyage code model (AI, 2024) and OpenAI's embeddings (Neelakantan et al., 2022); however, both are closed-source models, limiting their accessibility and adaptability for the wider research community.

4.2 Code Retrieval Augmented Generation

Neural code generation has been an important task (Lu et al.), and increasingly strong code language models have been developed (Roziere et al., 2023; Nijkamp et al., 2023; Li et al., 2023a; Guo et al., 2024; Team, 2024) to solve various tasks (Chen et al., 2021; Lai et al., 2023; Jimenez et al., 2024). However, most LMs generate code solely from natural language problem descriptions and the models' parametric knowledge, without leveraging external programming resources or using a retrieval-augmented generation approach. While prior work has focused on text-centric tasks with general-domain corpora like Wikipedia (Asai et al., 2024), some studies have used retrieved programming context from repositories (Ding et al., 2024; Yang et al., 2024) or documentation (Zhou et al., 2023). Code retrieval is crucial for enhancing code generation in RAG systems because it allows models to access relevant external code resources, leading to more accurate and context-aware code outputs. Our code retrieval model demonstrates significant improvements in code RAG performance, underscoring the importance of effective code retrieval in code generation tasks.

5 Conclusion

In the underexplored field of code retrieval, we introduce CODEXEMBED, a family of code embedding models ranging from 400M to 7B parameters. Our training pipeline unifies multiple programming languages and converts diverse code-related tasks into a common retrieval framework. Our models surpass the previous SOTA code retrieval by over 20% on the CoIR benchmark and achieve performance comparable to SOTA text retrievals on the BEIR benchmark. By enhancing retrieval capabilities, we demonstrate significant improvements in end-to-end retrieval-augmented generation for code-related tasks. By bridging the gap between text and code retrieval domains and releasing our models to the community, we aim to promote further research and innovation in developer tools and programming language understanding.

6 Limitations

While CODEXEMBED achieves state-of-the-art results in code and text retrieval, there are certain limitations to consider. First, the model's large size, particularly the 7B parameter variant, leads to significant storage and computational costs. With embeddings of 4096 dimensions, the model is more resource-intensive compared to smaller or more efficient representations. Although embedding size reduction techniques, such as Matryoshka representation learning (Kusupati et al., 2022), have shown promise in maintaining performance with lower dimensions, further research is needed to apply these approaches effectively to code retrieval without compromising performance. Additionally, while CODEXEMBED generalizes well across different programming languages and text retrieval tasks, it may still struggle with niche programming domains or highly specialized languages that were underrepresented in the training data. Future work could explore adapting the model for these underrepresented languages and improving efficiency without sacrificing accuracy.

References

Voyage AI. 2024. Voyage code 2: Elevate your code retrieval. https://blog.voyageai.com/2024/01/23/voyage-code-2-elevate-your-code-retrieval/. [Online; accessed 8-October-2024].

Emily Judith Arteaga Garcia, João Felipe Nicolaci Pimentel, Zixuan Feng, Marco Gerosa, Igor Steinmacher, and Anita Sarma. 2024. How to support ml end-user programmers through a conversational agent. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pages 1–12.

Akari Asai, Zexuan Zhong, Danqi Chen, Pang Wei Koh, Luke Zettlemoyer, Hannaneh Hajishirzi, and Wen-tau Yih. 2024. Reliable, adaptable, and attributable language models with retrieval. *arXiv preprint arXiv:2403.03187*.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, et al. 2016. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*.

Md Mustakim Billah, Palash Ranjan Roy, Zadia Codabux, and Banani Roy. 2024. Are large language models a threat to programming platforms? an exploratory study. *arXiv preprint arXiv:2409.05824*.

Deng Cai, Yan Wang, Lemao Liu, and Shuming Shi. 2022. Recent advances in retrieval-augmented text generation. In *Proceedings of the 45th international ACM SIGIR conference on research and development in information retrieval*, pages 3417–3419.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 2318–2335.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Xin Cheng, Di Luo, Xiuying Chen, Lemao Liu, Dongyan Zhao, and Rui Yan. 2024. Lift yourself up: Retrieval-augmented text generation with self-memory. *Advances in Neural Information Processing Systems*, 36.

Alessandro Del Sole. 2021. Introducing visual studio code. In *Visual Studio Code Distilled: Evolved Code Editing for Windows, macOS, and Linux*, pages 1–15. Springer.

Yangruibo Ding, Zijian Wang, Wasi Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, et al. 2024. Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion. *Advances in Neural Information Processing Systems*, 36.

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

Catherine Finegan-Dollak, Jonathan K Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-sql evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360.

Luyu Gao, Yunyi Zhang, Jiawei Han, and Jamie Callan. 2021. Scaling deep contrastive learning batch size under memory limited setup. In *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP-2021)*, pages 316–321.

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.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, YK Li, et al. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2022. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

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*.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. Towards unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*, 2(3).

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2023. Unsupervised dense information retrieval with contrastive learning. *Transactions on Machine Learning Research*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

Hyoungwook Jin, Seonghee Lee, Hyungyu Shin, and Juho Kim. 2024. Teach ai how to code: Using large language models as teachable agents for programming education. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–28.

Md Eusha Kadir, Tasnim Rahman, Sourav Barman, and Md Al-Amin. 2024. Exploring the competency of chatgpt in solving competitive programming challenges. *International Journal*, 13(1).

Rabimba Karanjai, Lei Xu, and Weidong Shi. 2024. Solmover: Smart contract code translation based on concepts. In *Proceedings of the 1st ACM International Conference on AI-Powered Software*, pages 112–121.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1, page 2. Minneapolis, Minnesota.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, et al. 2022. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35:30233–30249.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2023. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pages 18319–18345. PMLR.

Chankyu Lee, Rajarshi Roy, Mengyao Xu, Jonathan Raiman, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. 2024. Nv-embed: Improved techniques for training llms as generalist embedding models. *arXiv preprint arXiv:2405.17428*.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024a. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023a. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.

Xiangyang Li, Kuicai Dong, Yi Quan Lee, Wei Xia, Yichun Yin, Hao Zhang, Yong Liu, Yasheng Wang, and Ruiming Tang. 2024b. Coir: A comprehensive benchmark for code information retrieval models. *arXiv preprint arXiv:2407.02883*.

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

Shangqing Liu, Yu Chen, Xiaofei Xie, Jing Kai Siow, and Yang Liu. Retrieval-augmented generation for code summarization via hybrid gnn. In *International Conference on Learning Representations*.

Tianyang Liu, Canwen Xu, and Julian McAuley. 2024. Repobench: Benchmarking repository-level code auto-completion systems. In *The Twelfth International Conference on Learning Representations*.

Ye Liu, Kazuma Hashimoto, Yingbo Zhou, Semih Yavuz, Caiming Xiong, and S Yu Philip. 2021. Dense hierarchical retrieval for open-domain question answering. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 188–200.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. 2024. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. Codexglue: A machine learning benchmark dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Jorge Martinez-Gil. 2024. Source code clone detection using unsupervised similarity measures. In *International Conference on Software Quality*, pages 21–37. Springer.

Rui Meng*, Ye Liu*, Shafiq Rayhan Joty, Caiming Xiong, Yingbo Zhou, and Semih Yavuz. 2024. Sfr-embedding-2: Advanced text embedding with multi-stage training.

Rui Meng, Ye Liu, Shafiq Rayhan Joty, Caiming Xiong, Yingbo Zhou, and Semih Yavuz. 2024. Sfr-embedding-mistral: Enhance text retrieval with transfer learning. Salesforce AI Research Blog.

Gabriel de Souza P Moreira, Radek Osmulski, Mengyao Xu, Ronay Ak, Benedikt Schifferer, and Even Oldridge. 2024. Nv-retriever: Improving text embedding models with effective hard-negative mining. *arXiv preprint arXiv:2407.15831*.

Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, et al. 2022. Text and code embeddings by contrastive pre-training. *arXiv preprint arXiv:2201.10005*.

Erik Nijkamp, Hiroaki Hayashi, Caiming Xiong, Silvio Savarese, and Yingbo Zhou. 2023. Codegen2: Lessons for training llms on programming and natural languages. *arXiv preprint arXiv:2305.02309*.

Rangeet Pan, Ali Reza Ibrahimzada, Rahul Krishna, Divya Sankar, Lambert Pouguem Wassi, Michele Merler, Boris Sobolev, Raju Pavuluri, Saurabh Sinha, and Reyhaneh Jabbarvand. 2024. Lost in translation: A study of bugs introduced by large language models while translating code. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13.

Md Rizwan Parvez, Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Retrieval augmented code generation and summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2719–2734.

Huy N Phan, Hoang N Phan, Tien N Nguyen, and Nghi DQ Bui. 2024. Repohyper: Better context retrieval is all you need for repository-level code completion. *arXiv preprint arXiv:2403.06095*.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Ankita Nandkishor Sontakke, Manasi Patwardhan, Lovekesh Vig, Raveendra Kumar Medicherla, Ravindra Naik, and Gautam Shroff. 2022. Code summarization: Do transformers really understand code? In *Deep Learning for Code Workshop*.

Weisong Sun, Chunrong Fang, Yuchen Chen, Quanjun Zhang, Guanhong Tao, Yudu You, Tingxu Han, Yifei Ge, Yuling Hu, Bin Luo, et al. 2024. An extractive-and-abstractive framework for source code summarization. *ACM Transactions on Software Engineering and Methodology*, 33(3):1–39.

CodeGemma Team. 2024. Codegemma: Open code models based on gemma. *arXiv preprint arXiv:2406.11409*.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. Beir: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

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. *arXiv preprint arXiv:2212.03533*.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2023. Improving text embeddings with large language models. *arXiv preprint arXiv:2401.00368*.

Yining Wang, Liwei Wang, Yuanzhi Li, Di He, and Tie-Yan Liu. 2013. A theoretical analysis of ndcg type ranking measures. In *Conference on learning theory*, pages 25–54. PMLR.

Zora Zhiruo Wang, Akari Asai, Xinyan Velocity Yu, Frank F Xu, Yiqing Xie, Graham Neubig, and Daniel Fried. 2024. Coderag-bench: Can retrieval augment code generation? *arXiv preprint arXiv:2406.14497*.

Michel Wermelinger. 2023. Using github copilot to solve simple programming problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 172–178.

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

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. 2024. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th International ACM SIGIR*

Conference on Research and Development in Information Retrieval, pages 641–649.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*.

Burak Yetiştiren, Işık Özsoy, Miray Ayerdem, and Eray Tüzün. 2023. Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt. *arXiv preprint arXiv:2304.10778*.

Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. 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.

Shuyan Zhou, Uri Alon, Frank F Xu, Zhengbao Jiang, and Graham Neubig. 2023. Docprompting: Generating code by retrieving the docs. In *The Eleventh International Conference on Learning Representations*.

A Appendix

A.1 Dataset Details

Our training dataset contains a total of 3.36M training pairs, covering 12 different programming languages. As shown in Figure 3, the distribution of programming languages is imbalanced, with the majority of the data concentrated in a few popular languages. Python represents the largest portion of the dataset at 27.1%, followed by Go with 25.2%, and JavaScript and PHP at 17.0% and 17.2%, respectively. The remaining languages, including Java, SQL, Ruby, and others, account for smaller proportions, with Rust, Kotlin, and C# making up the smallest shares of the dataset.

A.2 Implementation Details

We summarize the hyperparameters in Table 8. For the code training data, we prepend a prompt to the query in the format: “Instruct: Given Code or Text, retrieve relevant content. Query:”.

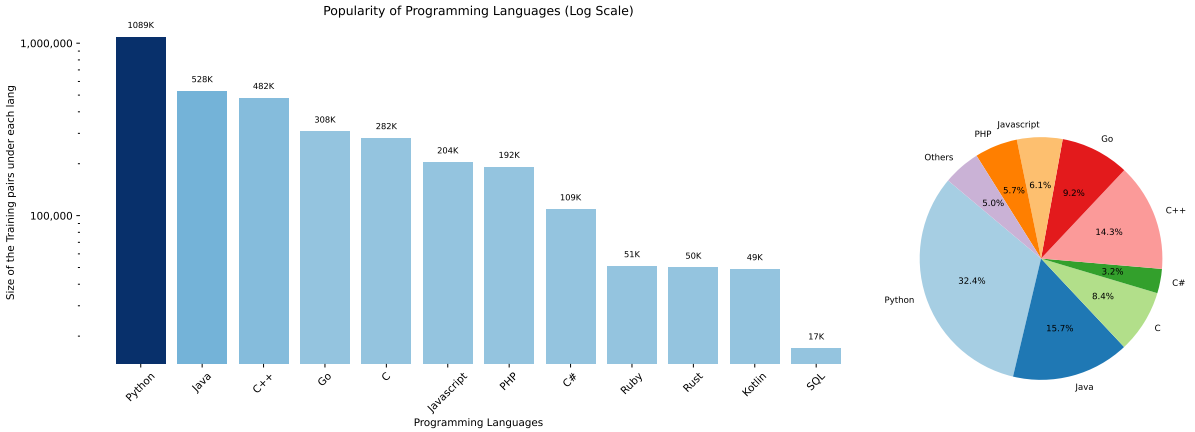

Figure 3: The programming language distribution of code training data in the general training stage.

	CodeXEmbed$_{400M}$	**CodeXEmbed**$_{2B}$	**CodeXEmbed**$_{7B}$
base model	`gte-large-en-v1.5`	`gemma-2-2b-it`	`Mistral-7B-Instruct-v0.3`
max learning rate	5×10^{-5}	5×10^{-5}	5×10^{-5}
GradCache	4	16	32
tuning parameters	Fully Model	LoRA	LoRA
Lora Rank	-	8	8
warmup steps	100	50	50
batch size	1024	1024	1024
max length	512	512	512
weight decay	0.01	0.01	0.01
hard negatives	7	7	7
bidirectional attention	✓	✓	✓
text:code ratio	1:1	1:3	1:3
pooling	bos	eos	eos

Table 8: Hyperparameters for contrastive code and text training