ProCQA: A Large-scale Community-based Programming Question Answering Dataset for Code Search

Zehan Li[1,2], **Jianfei Zhang**[3], **Chuantao Yin**[2], **Yuanxin Ouyang**[3], **Wenge Rong**[1,3]
[1]State Key Laboratory of Complex & Critical Software Environment, Beihang University, China
[2]Sino-French Engineer School, Beihang University, China
[3]School of Computer Science and Engineering, Beihang University, China
{lizehan, zhangjf, chuantao.yin, oyyx, w.rong}@buaa.edu.cn

Abstract

Retrieval-based code question answering seeks to match user queries in natural language to relevant code snippets. Previous approaches typically rely on pretraining models using crafted bi-modal and uni-modal datasets to align text and code representations. In this paper, we introduce ProCQA, a large-scale programming question answering dataset extracted from the StackOverflow community, offering naturally structured mixed-modal QA pairs. To validate its effectiveness, we propose a modality-agnostic contrastive pre-training approach to improve the alignment of text and code representations of current code language models. Compared to previous models that primarily employ bimodal and unimodal pairs extracted from CodeSearchNet for pre-training, our model exhibits significant performance improvements across a wide range of code retrieval benchmarks.

Keywords: Code QA Dataset, Code Search, Contrastive Pretraining

1. Introduction

Code Question Answering (Code QA) represents a pivotal research area in software intelligence. One popular task formulation is retrieval-based QA (Gu et al., 2018), in which the primary objective is to effectively match user queries expressed in natural language to relevant code snippets from an existing corpus. The prevailing approach for retrieval-based Code QA has been the utilization of dual-encoder-based representation models. The core idea underlying this approach is to map natural language queries and code snippets into a shared representation space, where closely located vectors correspond to semantically similar meanings.

To learn a shared representation space for text and code, early research efforts adopted masked language modeling (MLM) objective on paired text-code dataset to align different modalities (Kanade et al., 2020; Feng et al., 2020), similar to the monolingual and cross-lingual pre-training approaches (Devlin et al., 2019; Conneau and Lample, 2019). Subsequent work discovered the potential of contrastive pre-training, and applied it to code representation learning by constructing large-scale paired datasets (Jain et al., 2021; Li et al., 2022).

Current contrastive code representation learning methods such as CodeRetriever (Li et al., 2022) typically rely on curated uni-modal (code-code pairs) or bi-modal data (text-code pairs). Few work even uses distinct encoder for text and code (Heyman and Cutsem, 2020; Salza et al., 2023). Such pre-training design emphasizes the concept of modality distinction, diverging from the goal of establishing a unified representation space for different modalities. In Figure 1, we illustrate different data formats used for contrastive pre-training and analyse their chunk-level matching patterns. Uni-modal data offers code-code matching patterns, whereas bi-modal data implies code-text matching patterns. While a combination of both data types during pre-training can enable models to learn both matching signals, a more data-efficient approach to capture all matching patterns is through mixed-modal data.

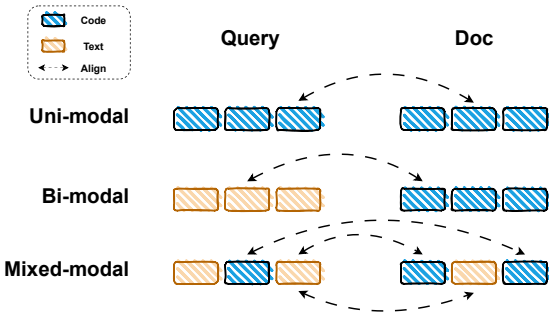

Figure 1: Illustration of different data formats used for contrastive representation alignment. Color represents chunk modality. Unimodal data focuses on code-to-code matching, while bimodal data emphasizes cross-modal matching. The mixed-modal data in ProCQA enables simultaneous learning of all matching patterns.

Besides, the majority of code embedding models (Jain et al., 2021; Wang et al., 2021a; Li et al., 2022) have primarily relied on CodeSearchNet (Husain et al., 2019) as the main pre-training corpus. While CodeSearchNet is a valuable resource, its size and data distribution have inherent limita-

LREC-COLING 2024, pages 13057–13067

Question	Segmentation fault while copying a string to the memory allocated array
Description	Following is a program I am practicing; int main() int i = 0; char **grid = (char **) malloc(5*sizeof(int)); for (i = 0 ; i < 5 ; i++) grid[i] = (char *) malloc(6); strcpy(grid[0], "eabcd"); strcpy(grid[1], "fghij"); strcpy(grid[2], "olkmn"); strcpy(grid[3], "trpqs"); strcpy(grid[4], "xywuv"); /*Segmentation Fault at this line*/ return 0; I am getting a segmentation fault at the line strcpy(grid[4], "xywuv"); . What could be the reason? I have allocated the array to have 5 strings(rows) of 6 characters each(columns).
Answer	You are allocating the wrong type at line 3 char **grid = (char **) malloc(5*sizeof(int)); Should be char **grid = (char **) malloc(5*sizeof(char*)); This is because you are declaring string-array. Therefore, the malloc should be char* (string / character pointer) Also the same if you were trying to declare 2-D integer array. It will be int **grid = (int **) malloc(5*sizeof(int*));

Table 1: An example sampled from the C programming language subset of ProCQA. Text and code are interleaved in these QA pairs.

tions that may impact the quality and diversity of learned code representations. Recent work has proposed to curate large-scale code datasets from GitHub (Allal et al., 2023). Yet their efforts mainly focus on training large-scale generative language models (LMs). In parallel, some research endeavors have aimed to create code-related question-answering datasets from diverse sources, as evidenced by Huang et al. (2021); Lee et al. (2022). Nevertheless, most of these datasets remain constrained by their scale, rendering them more suitable for stand-alone evaluation benchmarks rather than comprehensive pre-training corpus.

Therefore in this research, we try to bridge these gaps by proposing **ProCQA**, a large-scale community-based programming question answering dataset mined from StackOverflow. ProCQA encompasses an extensive collection of approximately 5 million QA pairs, spanning 11 different programming languages. This dataset is distinguished by its comprehensive language coverage, the diversity of user queries, and its code-mixing data format[1]. It can be used as both an evaluation benchmark and a pre-training corpus. We provide strict rule-based filtering and data decontamination procedure to ensure its quality and fairness. Different types of baseline models are trained and compared on this dataset to test its suitability as an evaluation benchmark.

To assess the efficacy of our proposed dataset as a pre-training corpus, we conducted large-scale modality-agnostic contrastive pretraining (MACP) on the code-mixing dataset, without making distinctions between text and code modalities. To demonstrate whether MACP can learn a better aligned representation space, we evaluated it on extensive code retrieval benchmarks, covering supervised, zero-shot, and out-of-domain scenarios. Experiments reveal that compared to previous pre-trained code language models, MACP achieves substantial improvements on most tasks we considered, advancing the previous best code retrieval model CodeRetriever (Li et al., 2022) by 1$\sim$10% points across different evaluation benchmarks. Comprehensive ablation and analysis demonstrates the effectiveness of our proposed approach.

The contributions of this paper can be summarized as follows:

- We create ProCQA, a large-scale dataset for programming question answering. ProCQA is characterized by its practicality, diversity and mixed-modal data format. We demonstrate its potential as an evaluation benchmark for comparing different code language models.
- Based on ProCQA, we present MACP, a code representation model pre-trained with modality-agnostic contrastive learning on the large-scale code-mixing dataset. MACP demonstrates remarkable performance gains over prior approaches across a wide range of code retrieval tasks.

2. Related Work

2.1. Code QA

Code-based question answering is a sub-problem of question answering. Different from the generative formulation, retrieval-based code QA aims to retrieve the most similar code from a large-scale code corpus, satisfying user requests. To evaluate the neural code search ability of current models, CodeSearchNet (Husain et al., 2019) was constructed by mining large-scale comment-code pairs from public GitHub repositories. Additionally, to evaluate the code comprehension ability of language models, Liu and Wan (2021) introduced CodeQA, a free-form code question-answering

[1]Please refer to Table 1 for an illustrative example.

dataset. This dataset was derived from existing code summarization datasets mined from GitHub, including two widely-used programming languages Python and Java. CodeQA synthesizes various types of question-answer pairs from code comments and documentation strings using manually curated rules, templates, and a range of NLP toolkits.

Recent work has been focused on constructing code QA dataset from real-world scenarios. For example, CoSQA (Huang et al., 2021) mines real-world user queries from Bing search logs and utilizes models trained on CodeSearchNet and humans to label corresponding code. Moreover, educational programming QA datasets have also gained attention. CS1QA (Lee et al., 2022) collects student questions and answers from teaching assistants on an online forum designed for an introductory Python programming course. This dataset offers insights into the educational applications of code-based question answering.

2.2. Code Language Models

Language models pre-trained on large-scale unlabeled corpora have demonstrated significant potential in code understanding and generation tasks. Prior works such as CodeBERT (Feng et al., 2020) employed replaced language modeling on unimodal and bi-modal data for pre-training. GraphCodeBERT (Guo et al., 2021) advanced this approach by harnessing data flow encoded in the Abstract Syntax Tree (AST) of code to enrich code structural information during pre-training. UniXCoder (Guo et al., 2022) unified three pre-training designs into one architecture and utilized AST structure and code comment to enhance the cross-modal alignment. There are also some work on adapting generative language models for code, as exemplified by CodeT5 (Wang et al., 2021b) and PLBART (Ahmad et al., 2021). These models incorporate code structure information into the design of specific pre-training tasks. Contrastive methods have also been introduced into code pre-training by several recent works with different approaches proposed for constructing positive and negative pairs (Jain et al., 2021; Wang et al., 2021a; Ding et al., 2022; Bui et al., 2021; Li et al., 2022).

It is worth noting that current code language models' pre-training corpus are primarily sourced from CodeSearchNet, consisting of 2 million code-text pairs. Limited efforts have been dedicated to mining large-scale datasets from GitHub (Allal et al., 2023; Li et al., 2023), but they mainly focus on training decoder language models rather than code representation models. An exception is the work by OpenAI (Neelakantan et al., 2022), but their models are only available via paid APIs and training data is not detailed.

3. ProCQA

In this section, we outline the methodologies employed in the creation of the ProCQA dataset, along with the filtering strategies applied to ensure data quality and fairness. Additionally, we present an analysis of various dataset statistics and define two tasks utilizing this dataset to evaluate different baseline models. The source code is available at https://github.com/jordane95/procqa.

3.1. Data Acquisition

To ensure the diversity and reflect real world user problems, we crawl our dataset from StackOverflow, a question answering community focusing on solving programming problems. Users can post their problems on the website and wait for others' answers. One characteristic of this dataset is that both the question and answer are code-mixing, i.e., text and code are interleaved within these fields. Such data format is very useful to indoctrinate and evaluate the model's matching ability of different patterns.

We use the public dumps as of 2022/12 for raw data downloading[2]. We extract the textual content consisting of code and texts from XML files. Three fields (title, question, answer) are kept. HTML tags are removed and only text content are kept using BeautifulSoup library.

3.2. Data Cleaning

A critical problem with these QA communities is that there are many unanswered questions and wrong answers. To handle this issue, we apply some rule-based approaches to filter out low-quality questions and answers.

More specifically, we filter out questions/answers that are either too short (< 20 characters) or too long (> 4096 characters). We only keep questions that have answers marked as accepted by the questioner since it is a natural annotation signal indicating the answer is helpful for the user.

3.3. Data Format

Data in ProCQA is formatted as triples illustrated in Table 1. The question is a concise user request. It is coupled with a detailed description which explains the problem in more detail. The answer is posted by other user and is the one accepted by the questioner. Note that in all data fields, code and text are interleaved, which provides a natural supervision signal for aligning the two modalities.

PL	C	C++	Java	Python	Ruby	Lisp	JavaScript	C#	Go	Rust	PHP
Size	204746	418346	831697	1008478	131218	4612	1217095	817970	36011	15514	567357

Table 2: Number of QA pairs for each programming language in ProCQA.

3.4. Data Statistics

We partition the dataset into different programming language subsets according to their tags contained in meta information. We consider the following eleven languages based on their popularity: Python, Java, JavaScript, Ruby, C, C++, C#, Rust, PHP, Lisp and Go. Dataset statistics are shown in Table 2. We split the dataset into train / valid / test set by a proportion of 80%:10%:10% following chronological order of posting date.

In addition, we analyse the question and answer length distribution of our ProCQA dataset in Figure 2. Most of the QA pairs in ProCQA contain dozens or hundreds of words, which are much closer to real user questions.

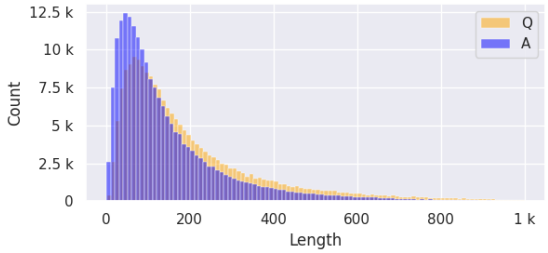

Figure 2: Question and answer length distribution in ProCQA (C subset).

3.5. Decontamination

Since ProCQA is crawled from StackOverflow, it many overlap with some evaluation sets constructed from the same source. To avoid data contamination, we perform evaluation data deduplication for our ProCQA training set.

Specifically, we employ two methods for deduplication. The first one is based on substring matching. Training example in ProCQA dataset is dropped if it contains any substring that is part of the queries in the evaluation set. We use three evaluation sets to perform deduplication (CoNaLa, SO-DS and StaQC). After this step, about 0.5% examples from the Python subset are dropped. Other subsets are influenced lightly. We also apply fuzzy deduplication method based on MinHash but no additional duplicate is found.

3.6. Comparison to previous datasets

To better understand the difference with previous dataset, we summarize some key factors of our ProCQA and previous ones in Table 3, including the number of supported programming languages (PLs), data format, size and data source.

CodeNN (Iyer et al., 2016) is also a dataset mined from StackOverflow for code summarization but contains much smaller amount of training examples and languages. CodeSearchNet (CSN) is on pair with ProCQA in terms of languages and size but drawn from a different data distribution (GitHub). Its queries are either documentation strings or comments rather than natural language questions, limiting its practicality in real scenarios. CoSQA and CS1QA contain some real user queries collected from Bing logs and classrooms but only cover Python and are limited in size.

In summary, ProCQA differs from previous work in the following main aspects:

1. More diverse language distribution at a larger scale.

2. Long-form questions and answers more aligned with real-world scenarios.

3.7. Tasks

We define two tasks based on the collected dataset for pilot exploration, including answer retrieval and generation. We choose C subset as a test bed for comparing multiple language models.

Answer Retrieval This task is defined as finding the correct answer from a large-scale answer corpus. We use answers from all splits of the dataset to form retrieval corpus. The query is the concatenation of question and description. We choose BM25 and some recent neural language models, such as BERT (Devlin et al., 2019), CodeBERT (Feng et al., 2020) and UniXCoder (Guo et al., 2022). Neural LMs are fine-tuned with the contrastive learning objective (*i.e.*, InfoNCE loss) on the question answer pairs from the training set. All models are trained for 3 epochs with the batch size of 32 and the learning rate of 2e-5. Both questions and answers are truncated to be maximum of 256 tokens. We choose MRR@10, Recall@10 and Recall@100 as main evaluation metrics.

Results are demonstrated in Table 4. We observe that text-only language models such as

[2]https://archive.org/details/stackexchange

Dataset	# of PLs	Data Format	Size	Data Source
CodeNN	2	Title, code	∼187K pairs	StackOverflow
CodeSearchNet	6	Comment, code	∼2M pairs	GitHub
CodeQA	2	Question, answer, code	∼190K pairs	GitHub
CoSQA	1	Query, code	∼20K pairs	Web search
CS1QA	1	Chat log, question, answer, type, code	∼9K pairs	Classroom
ProCQA	11	Question, description, answer	∼5M pairs	StackOverflow

Table 3: Comparison between different code-based datasets.

Model	MRR@10	R@10	R@100
BM25	51.7	61.1	73.1
BERT	48.3	62.0	79.7
CodeBERT	53.0	66.8	83.5
UniXCoder	58.4	71.8	86.1

Table 4: Answer retrieval performance of different language models on the C subset of ProCQA.

BERT are even inferior to unsupervised BM25, in terms of MRR@10. With code-specific pre-training, CodeBERT can outperform the strong BM25 baseline. More recent code language models such as UniXCoder performs best on this task.

Answer Generation We also consider a generative task formulation, in which the model is required to directly generate the answer to the question without additional reference. Similarly, we benchmark several generative language models on this task. Selected baseline models include T5 (Raffel et al., 2020), CodeT5 (Wang et al., 2021b), PLBART (Ahmad et al., 2021). Models are trained in a sequence-to-sequence manner by optimizing the cross-entropy loss of the answer sequence given question sequence with the same training hyperparameters as stated above. During inference, beam search decoding is used with a beam size of 5. We use ROUGE (Lin, 2004) as main evaluation metrics for this task and demonstrate results in Table 5.

Model	ROUGE_1	ROUGE_2	ROUGE_L
T5	14.3	2.6	11.8
CodeT5	17.6	4.8	14.0
PLBART	19.9	5.9	15.3

Table 5: Answer generation results of different baselines on ProCQA (C subset).

It is found that code language models is better than text-only models, indicating the effectiveness of code-specific pre-training. Even the best model struggles on this task because the answers are relatively long (mostly 100-200 words, see Figure 2). This indicates that ProCQA is a challenging dataset for long-form generative QA task. How to improve the long-form question-answering performance of language models with limited parameters is also an interesting direction for future research.

4. Experiments

To assess the quality and utility of our proposed dataset, we evaluate its benefits to other code search benchmarks when acting as a pre-training corpus. We also conduct ablation experiments to demonstrate the effectiveness of ProCQA over existing pre-training corpus CSN.

4.1. Settings

Our model basically follows the two-tower architecture in which vector representations for code and text are produced by mean pooling over the last layer hidden representations of the language models. It is trained via the contrastive objective using the InfoNCE loss

$$\mathcal{L} = -\log \frac{e^{s(q,d)/\tau}}{e^{s(q,d)/\tau} + \sum\limits_{d' \in \mathcal{D}_-} e^{s(q,d')/\tau}}. \quad (1)$$

where q denotes the question, d denotes the corresponding answer, $\mathcal{D}_-$ is a set of negative samples, τ is the temperature. $\mathcal{D}_-$ is also enlarged with other examples from the same batch.

The main baselines we compare to are GraphCodeBERT (Guo et al., 2021) and CodeRetriever (Li et al., 2022). In addition to the text-code pairs in CSN used by GraphCodeBERT, CodeRetriever also employs sophisticated rules and learned models for mining high-quality code-code and code-text pairs from the raw CSN code corpus. Instead we use commonly available QA pairs from ProCQA mined by weak supervision. We use the training split across all languages to construct different types of mixed-modal positive pairs. We apply modality-agnostic contrastive pre-training on the ProCQA and CSN dataset and compare our model to previous code embedding models on various retrieval tasks. Our model is denoted as MACP.

4.2. Implementation Details

For fair comparison, our model is initialized with GraphCodeBERT, same as CodeRetriever. MACP is pre-trained with the contrastive objective in Equation 1 using cosine similarity and $\tau = 0.01$. To balance low-resource languages, we sample each data batch from a multi-nominal distribution over different language subsets

$$p_i = \frac{n_i^\alpha}{\sum_{j=1}^{n} n_j^\alpha}, \quad (2)$$

with n_i equal to the size of subset i and smoothing parameter $\alpha = 0.5$. We run the contrastive pre-training for 10k steps with a global batch size of 6192. In-batch negatives are used and shared across different GPUs. Each sequence is truncated at a maximum length of 128. The learning rate is initially warmed up to 2e-4 for the first 10% steps, followed by a linear decay.

We utilize the same contrastive loss during fine-tuning on each downstream dataset. Each fine-tuning experiment only involves one dataset so we directly sample data after shuffling it. Models are trained using a peak learning rate of 2e-5 with the same scheduler as pre-training. The maximum sequence length is 512. Batch size is 128 and each sample is accompanied with 7 randomly sampled negatives. Training epochs is 3. Other hyperparameters are same as pre-training. We only consider in-batch negatives for contrastive learning so we compare models under this setting.

We conduct all experiments on two NVIDIA A100 GPUs with 40G memory. We use DeepSpeed, gradient checkpointing and mixed precision (FP16) encoding to reduce memory cost. The pre-training process takes about 18 hours. Fine-tuning on all datasets is finished in one day.

4.3. Evaluation Benchmarks

To provide an extensive evaluation of the generalization ability of our pre-trained models, we select a large variety of code retrieval tasks from different domains under different settings.

We first evaluate on the CodeSearchNet benchmark (Husain et al., 2019), which is widely used for evaluating the text-code search ability of code retrieval models. One drawback of CodeSearchNet is the queries are not aligned to real user questions. So, we also evaluate on some more challenging datasets, Adv Test (Lu et al., 2021), CoSQA (Huang et al., 2021), CoNaLa (Yin et al., 2018), SO-DS (Heyman and Cutsem, 2020), StaQC (Yao et al., 2018). The last three evaluation datasets follow the setting of Heyman and Cutsem (2020), where during inference both text description and code snippet are used for matching. The main evaluation metric is MRR.

Then, code-code search results on POJ-104 (Mou et al., 2016) is also reported to evaluate the intra-modal retrieval ability. In this dataset, Python program solutions of the same problem is regarded as positive pairs. The objective is to retrieve relevant code snippets which answer the same problem. To investigate the cross-lingual code retrieval ability, we use CodeNet (Puri et al., 2021) as an evaluation benchmark, which is also a problem-solution dataset similar to POJ-104 but covers more languages. On CodeNet, we consider the zero-shot retrieval of three programming languages (Ruby, Python and Java) following Guo et al. (2022), where code is pre-processed by removing comments and replacing all separators with whitespace. Performance is evaluated by MAP.

Finally, to test whether our model can generalize to out-of-domain languages, we choose two text-code search datasets with languages unseen during pre-training. Smart Contracts (SC) (Yang et al., 2021) contains Solidity programming language and Spider (Yu et al., 2018) consists of SQL-query pairs. We use the dataset split released by Chai et al. (2022). Models are evaluated by Recall@{1,5,10} and MRR@1000. The statistics of downstream evaluation benchmarks are illustrated in Table 6.

Dataset	Lang	Train	Valid	Test
CSN	Ruby	24.9K	1.4K	1.3K
CSN	JS	58K	3.9K	3.3K
CSN	Go	167K	7.3K	8.1K
CSN	Python	252K	13.9K	14.9K
CSN	Java	165K	5.2K	10.9K
CSN	PHP	241K	13.0K	14.0K
Adv	Python	28.0K	9.6K	19.2K
CoSQA	Python	19K	0.5K	0.5K
CoNaLa	Python	2.8K	-	0.8K
SO-DS	Python	14.2K	0.9K	1.1K
StaQC	Python	20.4K	2.6K	2.7K
POJ104	Python	32K	8K	12K
CodeNet	Ruby	-	-	11.7K
CodeNet	Python	-	-	15.6K
CodeNet	Java	-	-	23.5K
SC	Solidity	57K	4.1K	1K
Spider	SQL	14K	2.1K	1K

Table 6: Statistics of downstream evaluation datasets.

4.4. Results

In this section, we report and discuss the performance of MACP on the evaluation benchmarks introduced in the previous section, spanning both supervised and zero-shot settings. In the supervised setting, MACP is directly fine-tuned on full training set and the last checkpoint is evaluated on

Method	Ruby	Javascript	Go	Python	Java	PHP	Overall
ContraCode (Jain et al., 2021)	-	30.6	-	-	-	-	-
SyncoBERT (Wang et al., 2021a)	72.2	67.7	91.3	72.4	72.3	67.8	74.0
CodeBERT (Feng et al., 2020)	67.9	62.0	88.2	67.2	67.6	62.8	69.3
GraphCodeBERT (Guo et al., 2021)	70.3	64.4	89.7	69.2	69.1	64.9	71.3
UniXcoder (Guo et al., 2022)	74.0	68.4	91.5	72.0	72.6	67.6	74.4
CodeRetriever (Li et al., 2022)	75.3	69.5	91.6	73.3	74.0	68.2	75.3
MACP	**77.8**	**72.5**	**92.4**	**76.1**	**75.7**	**70.1**	**77.4**

Table 7: MRR@1k on six programming language test sets of the CodeSearchNet.

Method	Adv	CoSQA	CoNaLa	SO-DS	StaQC	Overall
SyncoBERT (Wang et al., 2021a)	38.1	-	-	-	-	-
CodeBERT (Feng et al., 2020)	27.2	64.7	20.9	23.1	23.4	31.9
GraphCodeBERT (Guo et al., 2021)	35.2	67.5	23.5	25.3	23.8	35.1
UniXcoder (Guo et al., 2022)	41.3	70.1	-	-	-	-
CodeRetriever (Li et al., 2022)	**43.0**	70.6	29.6	27.1	**25.5**	39.0
MACP	39.7	**72.0**	**61.0**	**48.8**	23.3	**49.0**

Table 8: Text-code search performance (MRR@1k) on datasets that are closer to the real scenario.

the test set. In the zero-shot setting, it is directly evaluated on the test set.

Text-Code Search We first present evaluation results on six programming language subsets of CodeSearchNet in Table 7. MACP trained with the newly proposed dataset outperforms previous best model CodeRetriever on all language subsets, by an average of 2.1 points.

Next, we look at results on several challenging benchmarks, all collected from real-world user queries instead of docstrings. As shown in Table 8, our model significantly outperforms prior state-of-the-art models by up to 10 points on average. We attribute the improvement to real-world user queries from ProCQA.

Code-Code Search After evaluating the cross-modal search ability of our embedding model, we zoom into the intra-modal retrieval performance by evaluating on a code clone detection benchmark, POJ-104. Results are illustrated in Table 9. Our model outperforms previous best baseline CodeRetriever by +1.38 points.

Zero-Shot Cross-Lingual Code Search We list the cross-lingual code retrieval performance of our model MACP and other baselines from Guo et al. (2022) in Table 10. UniXCoder has significantly better zero-shot code retrieval performance, owing to its contrastive objective during pre-training. MACP consistently outperforms previous baselines by a large margin, setting new state-of-the-art performance on this task.

Method	MAP
RoBERTa (Liu et al., 2019)	76.67
CodeBERT (Feng et al., 2020)	82.67
GraphCodeBERT (Guo et al., 2021)	85.16
SynCoBERT (Wang et al., 2021a)	88.24
DISCO (Ding et al., 2022)	82.77
Corder (Bui et al., 2021)	84.10
CodeRetriever (Li et al., 2022)	88.85
MACP	**90.23**

Table 9: Performance of Python code-to-code retrieval task on POJ-104.

Cross-Domain Code Search In Table 11, we compare different models' performance on Solidity and SQL, two languages unseen during pre-training. Previous best model MAML (Chai et al., 2022) applied model-agnostic meta learning on CodeBERT (Feng et al., 2020) using Java and Python subsets from CSN for pre-training. In addition, we also report the performance of GraphCodeBERT using our codebase as another baseline for comparison. Our model significantly improves the cross-domain code search performance on unseen languages. One possible reason is that the diversity of language coverage in ProCQA equips the model with better language adaptation ability.

4.5. Analysis

Impact of pretraining data distribution We first investigate the effect of different pre-training corpus by doing a series of controlled experiments where only the pre-training data distribution is

Method	Ruby			Python			Java			Overall
	Ruby	Python	Java	Ruby	Python	Java	Ruby	Python	Java	
CodeBERT	13.55	3.18	0.71	3.12	14.39	0.96	0.55	0.42	7.62	4.94
GraphCodeBERT	17.01	9.29	6.38	5.01	19.34	6.92	1.77	3.50	13.31	9.17
PLBART	18.60	10.76	1.90	8.27	19.55	1.98	1.47	1.27	10.41	8.25
CodeT5	18.22	10.02	1.81	8.74	17.83	1.58	1.13	0.81	10.18	7.81
UniXcoder	29.05	26.36	15.16	23.96	30.15	15.07	13.61	14.53	16.12	20.45
MACP	**44.74**	**43.11**	**31.26**	**40.59**	**45.77**	**29.75**	**32.8**	**33.75**	**30.74**	**36.95**

Table 10: MAP score (%) of zero-shot code-to-code search task on CodeNet.

Method	Solidity				SQL			
	R@1	R@5	R@10	MRR	R@1	R@5	R@10	MRR
CodeBERT (Feng et al., 2020)	53.2	77.9	84.8	64.4	67.5	92.0	96.0	78.2
MAML (Chai et al., 2022)	65.8	82.9	87.9	73.4	74.6	95.2	97.2	83.7
GraphCodeBERT (Guo et al., 2021)	72.9	85.5	89.3	78.5	78.5	94.5	96.6	85.5
MACP	**75.2**	**87.3**	**90.6**	**80.7**	**85.4**	**96.0**	**97.4**	**90.3**

Table 11: Results of cross-domain code retrieval on programming languages unseen during pretraining.

changed. We run two additional experiments by using the CSN and ProCQA dataset individually for pre-training. Due to space limitation, we report downstream fine-tuned retrieval performance on CodeSearchNet in Figure 3. Results on other evaluation benchmarks follow the same trends.

Despite CSN belongs to in-domain data for this evaluation benchmark, it still underperforms ProCQA when being used as a pre-training corpus. Combining two datasets gives better results. This showcases the effectiveness of ProCQA dataset being used as a mixed-modal corpus for retrieval-oriented contrastive pre-training.

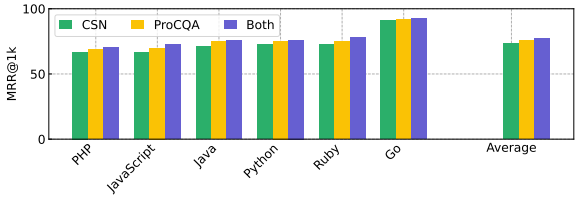

Figure 3: Ablation of the pre-training corpus. Results compared on test sets of CodeSearchNet.

Effect of modality-agnostic data We ablate on the choice of modality-agnostic contrastive learning by comparing to another setting which we explicitly distinguish text and code in data design. Due to the high difficulty of parsing incomplete code snippets in ProCQA, we conduct this ablation on the CSN pre-training corpus where code are well formed and can be parsed by existing tools. Bi-modal setting removes all comments in the code while mixed-modal setting keeps them. We list results in Table 12. The evaluation set Adv Test only requires code-text matching, yet training with mixed-modal data formats still has benefits.

Setting	Adv Test
bi-modal	38.8
mixed-modal	39.1

Table 12: Effect of the data format used in CodeSearchNet corpus pre-training. We report MRR@1k on Adv Test set.

Quantifying the effect of data contamination To avoid data contamination and ensure fairness, we performed de-duplication for the ProCQA dataset with respect to the relevant evaluation benchmarks from the same source, including CoNaLa, SO-DS and StaQC. In Table 13, we provide a quantitative analysis on the proportion of contaminated data for each evaluation set and the performance using raw and filtered version of the ProCQA dataset for pre-training. Although a large-proportion of the evaluation set is included in the raw pre-training data, removing them raises a limited degradation of model performance, as they only make up a small portion of the large-scale pre-training data.

Dataset	Proportion	Unfiltered	Filtered
CoNaLa	83.7%	62.9	61.0
SO-DS	48.4%	49.8	48.8
StaQC	31.4%	23.6	23.3

Table 13: Analysis on the influence of data contamination on three evaluation datasets.

5. Conclusion

In this work we introduce ProCQA, a large-scale community-based programming question answering dataset mined from StackOverflow with strict filtering strategies for quality and fairness control. ProCQA is featured by its practicality, diversity and code-mixing data format. Furthermore, through modality-agnostic contrastive pre-training on interleaved code and text data, our new dataset yields a language model that has a better aligned representation space between code and text, achieving state-of-the-art performance on a large spectrum of code retrieval tasks. In future work, it would be interesting to explore the benefit of ProCQA to other generative code QA tasks.

6. Acknowledgements

This work was supported by the National Natural Science Foundation of China (No.61977003) and the State Key Laboratory of Complex & Critical Software Environment (CCSE-2024ZX-16).

7. Bibliographical References

Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified pre-training for program understanding and generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2655–2668.

Loubna Ben Allal, Raymond Li, Denis Kocetkov, Chenghao Mou, Christopher Akiki, Carlos Muñoz Ferrandis, Niklas Muennighoff, Mayank Mishra, Alex Gu, Manan Dey, Logesh Kumar Umapathi, Carolyn Jane Anderson, Yangtian Zi, Joel Lamy-Poirier, Hailey Schoelkopf, Sergey Troshin, Dmitry Abulkhanov, Manuel Romero, Michael Lappert, Francesco De Toni, Bernardo García del Río, Qian Liu, Shamik Bose, Urvashi Bhattacharyya, Terry Yue Zhuo, Ian Yu, Paulo Villegas, Marco Zocca, Sourab Mangrulkar, David Lansky, Huu Nguyen, Danish Contractor, Luis Villa, Jia Li, Dzmitry Bahdanau, Yacine Jernite, Sean Hughes, Daniel Fried, Arjun Guha, Harm de Vries, and Leandro von Werra. 2023. Santacoder: Don't reach for the stars! *CoRR*, abs/2301.03988.

Nghi D. Q. Bui, Yijun Yu, and Lingxiao Jiang. 2021. Self-supervised contrastive learning for code retrieval and summarization via semantic-preserving transformations. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 511–521.

Yitian Chai, Hongyu Zhang, Beijun Shen, and Xiaodong Gu. 2022. Cross-domain deep code search with meta learning. In *Proceedings of the 44th IEEE/ACM International Conference on Software Engineering*, pages 487–498.

Alexis Conneau and Guillaume Lample. 2019. Cross-lingual language model pretraining. In *Proceedings of the 2019 Annual Conference on Neural Information Processing Systems*, pages 7057–7067.

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*, pages 4171–4186.

Yangruibo Ding, Luca Buratti, Saurabh Pujar, Alessandro Morari, Baishakhi Ray, and Saikat Chakraborty. 2022. Towards learning (dis)-similarity of source code from program contrasts. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, pages 6300–6312.

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.

Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *Proceedings of the 40th International Conference on Software Engineering*, pages 933–944.

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*, pages 7212–7225.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin B. Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. GraphCodeBERT: Pre-training code representations with data flow. In *Proceedings of the 9th International Conference on Learning Representations*.

Geert Heyman and Tom Van Cutsem. 2020. Neural code search revisited: Enhancing code snippet retrieval through natural language intent. *CoRR*, abs/2008.12193.

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*, pages 5690–5700.

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.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 2073–2083.

Paras Jain, Ajay Jain, Tianjun Zhang, Pieter Abbeel, Joseph Gonzalez, and Ion Stoica. 2021. Contrastive code representation learning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5954–5971.

Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. 2020. Learning and evaluating contextual embedding of source code. In *Proceedings of the 37th International Conference on Machine Learning*, pages 5110–5121.

Changyoon Lee, Yeon Seonwoo, and Alice Oh. 2022. CS1QA: A dataset for assisting code-based question answering in an introductory programming course. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2026–2040.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy V, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Moustafa-Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. StarCoder: May the source be with you! *CoRR*, abs/2305.06161.

Xiaonan Li, Yeyun Gong, Yelong Shen, Xipeng Qiu, Hang Zhang, Bolun Yao, Weizhen Qi, Daxin Jiang, Weizhu Chen, and Nan Duan. 2022. CodeRetriever: A large scale contrastive pre-training method for code search. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2898–2910.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Proceedings of ACL Workshop on Text Summarization Branches Out*, pages 74–81.

Chenxiao Liu and Xiaojun Wan. 2021. CodeQA: A question answering dataset for source code comprehension. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2618–2632.

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 pre-training 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. *CoRR*, abs/2102.04664.

Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. 2016. Convolutional neural networks over tree structures for programming language processing. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pages 1287–1293.

Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming

Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, Johannes Heidecke, Pranav Shyam, Boris Power, Tyna Eloundou Nekoul, Girish Sastry, Gretchen Krueger, David Schnurr, Felipe Petroski Such, Kenny Hsu, Madeleine Thompson, Tabarak Khan, Toki Sherbakov, Joanne Jang, Peter Welinder, and Lilian Weng. 2022. Text and code embeddings by contrastive pre-training. *CoRR*, abs/2201.10005.

Ruchir Puri, David S. Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir R. Choudhury, Lindsey Decker, Veronika Thost, Luca Buratti, Saurabh Pujar, Shyam Ramji, Ulrich Finkler, Susan Malaika, and Frederick Reiss. 2021. CodeNet: A large-scale AI for code dataset for learning a diversity of coding tasks. In *Proceedings of the 35th Annual Conference on Neural Information Processing Systems Track on Datasets and Benchmarks*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:140:1–140:67.

Pasquale Salza, Christoph Schwizer, Jian Gu, and Harald C. Gall. 2023. On the effectiveness of transfer learning for code search. *IEEE Transactions on Software Engineering*, 49(4):1804–1822.

Xin Wang, Yasheng Wang, Fei Mi, Pingyi Zhou, Yao Wan, Xiao Liu, Li Li, Hao Wu, Jin Liu, and Xin Jiang. 2021a. SynCoBERT: Syntax-guided multi-modal contrastive pre-training for code representation. *CoRR*, abs/2108.04556.

Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. 2021b. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8696–8708.

Zhen Yang, Jacky Keung, Xiao Yu, Xiaodong Gu, Zhengyuan Wei, Xiaoxue Ma, and Miao Zhang. 2021. A multi-modal transformer-based code summarization approach for smart contracts. In *Proceedings of the 29th IEEE/ACM International Conference on Program Comprehension*, pages 1–12.

Ziyu Yao, Daniel S. Weld, Wei-Peng Chen, and Huan Sun. 2018. StaQC: A systematically mined question-code dataset from stack overflow. In *Proceedings of the 2018 World Wide Web Conference*, pages 1693–1703.

Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to mine aligned code and natural language pairs from stack overflow. In *Proceedings of the 15th International Conference on Mining Software Repositories*, pages 476–486.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and Text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921.