A New Search Paradigm for Natural Language Code Search

Anonymous ACL submission

Abstract

Code search can accelerate the efficiency of software development by finding code snippets for the given query. The dominant code search paradigm is to learn the semantic matching between code snippets and queries by neural networks. However, this search paradigm causes the gap transferring and expansion between code snippets and queries because researchers utilize pairs of code snippets and code descriptions (e.g., comments and documentation) to train their models and evaluate the trained models on the query which is different from the code description in writing style and application scenario. To remedy the issue, we propose a new simple but effective search paradigm, Query2Desc, which entirely depends on natural language and conducts code search by performing the semantic matching between code descriptions and queries. Experimental results on dataset CoSQA show that the state-of-the-art model CodeBERT gets improvement of 17.48% in terms of the average MRR when applying it on Query2Desc. Moreover, baseline models on Query2Desc can return the right results in top-10 search results for at least 95% of queries in the test set of CoSQA.

1 Introduction

Natural language code search, a task that can return relevant code snippets when the user inputs a natural language query, is widely executed in various communities with programming requirements, e.g., software engineering, natural language processing (NLP), and computer vision (Allamanis et al., 2018; Liu et al., 2022a). To enable users to get satisfactory search results, a superior neural code search method is required to effectively measure the semantic similarity between a natural language query (henceforth referred to as query) and code snippets.

The current mainstream neural code search models are designed on the search paradigm Query2Code (Query to Code) (Gu et al., 2018; Wan et al., 2019; Shuai et al., 2020; Feng et al., 2020; Fang et al., 2021), which first embeds code snippets and queries into a unified vector space, then performs semantic matching for them in this vector space. In the training phase, however, these models are trained and verified on large-scale simulation datasets in which code descriptions (e.g., code comments or documentation (Gu et al., 2018; Husain et al., 2019)) are regarded as the query of a code snippet, which makes the gap transferring and expansion between code snippets and queries. There inherently is the gap between code snippets and code descriptions because of the variance of code and natural language[1] (Allamanis et al., 2015). When applying the trained models to the real-world scenario, the gap between code snippets and code descriptions is transferred to code snippets and queries. Besides, from Figure 1 we can observe that

Eample 1:
Query: 1d array in char datatype in python.
Comment: Convert Matrix attributes which are array-like or buffer to array.
Code:

```
def convert_to_array(array_like, dtype):
  if isinstance(array_like, bytes):
    return np.frombuffer(array_like,
      dtype=dtype)
  return np.asarray(array_like, dtype)
```

Eample 2:
Query: accessing a column from a matrix in python.
Comment: Return a column of the given matrix.
Code:

```
def get_column(self, X, column):
  if isinstance(X, pd.DataFrame):
    return X[column].values
  return X[:, column]
```

Figure 1: Two examples for the comment and the query of a code snippet, both of which are from CoSQA (Huang et al., 2021).

[1]Neural networks can bridge the gap but cannot eliminate the gap

there is not always a semantic consistency between the code description and the query of a snippet because code descriptions are written by developers for explaining the code function and queries are written by users for searching query-related code snippets. As the result, the above gap is also expanded in the real-world scenario. Therefore, previous neural models cannot perform as well as the validation stage in the real-world scenario.

To address the aforementioned issues, we propose Query2Desc (Query to Description), a new search paradigm that conducts code search by measuring the semantic similarity between code descriptions and queries. In this situation, we can regard code descriptions as the index of its corresponding code snippets. Code search models on Query2Desc only need to search similar code descriptions for a given query. We therefore transform the problem that learns the semantic matching between natural language and code into another problem, that is, to measure the semantic similarity of two natural language sentences, which is a simpler problem and solves the gap transferring and expansion.

We perform experiments on CodeSearchNet Challenge (Husain et al., 2019), Python_Q collected by us, and CoSQA (Huang et al., 2021). The experimental results show that the state-of-the-art model, CodeBERT, gets the improvement of 17.48% in terms of the average MRR when applying it on Query2Desc. We also find that simply combining Query2Desc with pre-trained models in NLP, e.g., BERT and RoBERTa, can also obtain the close performance with CodeBERT. Moreover, pre-trianed models on Query2Desc can return right results in top-10 search results for at least 95% of queries in the test data of CoSQA.

To sum up, we make the following contributions:

- We propose a new search paradigm Query2Desc for code search, which is entirely based on natural language. By using Query2Desc, we effectively eliminate the gap transferring and expansion between code snippets and queries.

- We conduct extensive experiments to explore the usefulness of Query2Desc and the evaluation results show that Query2Desc performs well on the code search task.

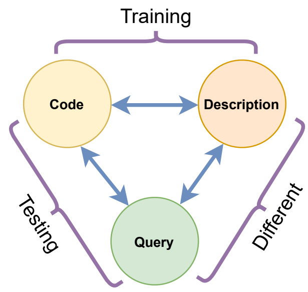

Figure 2: The relations between code snippets, code descriptions, and queries.

2 Background

In this section, we introduce the existing code search models and our motivation.

2.1 Code Search Models

Before considering deep learning technologies, most code search methods are based on information retrieval (IR) (Bajracharya et al., 2006; Lv et al., 2015; Lu et al., 2015; Nie et al., 2016; Rahman et al., 2019; Rahman, 2019; Liu et al., 2022b). These methods mainly depend on matching keywords in the query with code snippets to implement code search. Especially, some of them design methods to expand or reformulate the query for more accurate matching. Different from IR based models, deep learning based models learn contextual representations for code snippets and queries, representing them as low-dimensional dense vectors, then calculate their semantic similarity (e.g., cosine similarity) and return code snippets with the highest similarity scores (Gu et al., 2018; Shuai et al., 2020; Fang et al., 2021). Except for using text information to learn contextual representations for code snippets, some studies utilize the structural information of code snippets to learn their representation (Wan et al., 2019; Haldar et al., 2020; Guo et al., 2020). Although IR and deep learning are technically different, the above-mentioned code search models use the same search paradigm, that is, Query2Code.

There is another type of code search task, code-to-code search (Kim et al., 2018; Zhou et al., 2019). Since it focuses on semantic matching of programming languages, which is different with query-based code search while the former is generally towards the experienced developers and the latter is usually towards the novice developers.

2.2 Motivation

Generation of Gap To conduct code search on the previous search paradigm, Query2Code, neural models need to learn the semantic similarity between the code snippet and its corresponding query. Since source code is highly structured data (Hu et al., 2018; Shiv and Quirk, 2019), however, neural models cannot learn the representation for source code as effectively as the learned representation of natural language. The reason is that regarding the source code as the sequence may loss its structural information (Alon et al., 2018, 2019). To the best of our knowledge, a good model essentially makes a code snippet and its corresponding query have the highest semantic similarity, but the fact is that CodeBERT cannot perform as well as SimCSE-(BERT/RoBERTa) which achieves the state-of-the-art result on semantic textual similarity task. This fact shows that there still exists the gap between code snippets and queries in previous models.

Gap Transferring and Expansion To train an effective neural model on Query2Code, researchers need to collect enough code-query pairs. Due to the difficulty of collecting real-world queries, however, in CoSQA collected by MSRA with help of more than 100 participants, it only contains about 20K effective pairs of queries and code snippets. To obtain sufficient data, researchers generally use code descriptions to simulate queries (Gu et al., 2018; Husain et al., 2019) because there are enough high-quality code projects with complete documentation and code comments. As shown in Figure 2, when finishing training on pairs of code snippets and code descriptions, researchers use real-world queries to evaluate the effectiveness of their trained models. The gap between code snippets and code descriptions is transferred to code snippets and queries. Moreover, this gap is further expanded since code descriptions and queries have different writing style and application scenario. Although Huang et al. (2021) proposed to fine-tune CodeBERT on pairs of code snippets and queries, it only can alleviate the gap rather than eliminate it.

Inspiration Inspired by the success of the pretrianed model on semantic textual similarity task, we make an interesting assumption: if we can transform Query2Code into a simpler search paradigm that relies purely on natural language, the above problems may be well solved. Since we only model natural language on this paradigm, the gap between code snippets and natural language is eliminated. We just need to make the code description and its corresponding query have the highest semantic similarity, which requires us to find a model that can learn effective contextual representation for natural language. Actually, any pre-trained language model trained on large-scale corpus can well represent natural language. The remaining problem is whether there is such a search paradigm that only relies on natural language data to conduct code search, which motivates us to find it.

3 Approach

In this section, we first introduce Query2Desc, a new search paradigm that conducts natural language code search by measuring the semantic similarity between queries and code descriptions. Afterward, we build an example model, QudeBERT (Query2Desc BERT), to describe how to use Query2Desc to code search.

3.1 Query2Desc

As previous studies (Gu et al., 2018; Husain et al., 2019) can use code descriptions to simulate queries for obtaining sufficient data, it demonstrates that the mapping between code snippets and code descriptions is reliable. Then we can step out of the previous mindset and use code descriptions for another purpose, for example, the index of a code snippet. By building this index, we can regard code descriptions as the unique label of code snippets. In this situation, we can implement the code search by searching code descriptions according to the query. The complete code search process we conceive is shown in Figure 3. Instead of directly searching code snippets according to the query, we search their descriptions. When a user inputs a query to the neural search engine, e.g., QudeBERT, it first searches for a group of code descriptions which have the highest semantic similarities with the inputted query, then transforms them to code snippets by the one-to-one mapping. By conducting the above process, we conduct code search without using source code, successfully transforming Query2Code to a new search paradigm that relies purely on natural language data, i.e., code descriptions and queries. We call the above search paradigm Query2Desc.

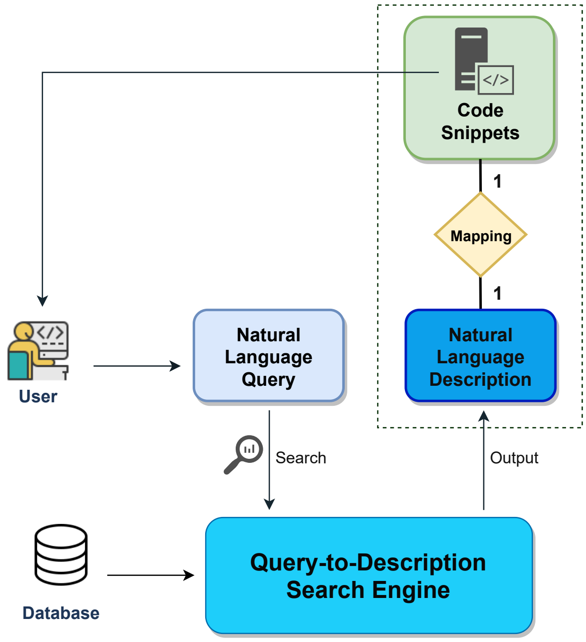

Figure 3: Query to Description Code Search Paradigm

3.2 QudeBERT

Model Architecture We follow BERT (Devlin et al., 2018), RoBERTa (Liu et al., 2019) and CodeBERT (Feng et al., 2020), and use multi-layer bidirectional Transformer (Vaswani et al., 2017) as the model architecture of QudeBERT. We construct QudeBERT by using exactly the same model architecture as BERT but we only use masked language model objective in the pre-training phase, which is the same with RoBERTa (Liu et al., 2019). We first initialize parameters of QudeBERT from BERT which was pre-trained on English Wikispedia and BooksCorpus. Then, we pre-trian QudeBERT on domain corpora composed of code descriptions. Finally, we conduct a two-stage fine-tuning for QudeBERT.

Input/Output Representations In the pre-training phase, we set the text sentence as a sequence of tokens with two special tokens, $[CLS]$ and $[EOS]$, thus the whole sentence can be expressed as $\{[CLS], w_1, ..., w_n, [EOS]\}$. In the fine-tuning phase, we concatenate paris of sentence A and sentence B and insert $[CLS]$ and $[SEP]$ tokens to each sentence, namely $\{[CLS], a_1, ..., [SEP]; [CLS], b_1, ..., [SEP]\}$. The output of QudeBert contains: 1) the representation of $[CLS]$, which is the aggregated representation for the whole sentence and can be used for some NLP tasks, such as sentiment analysis (Naseem et al., 2020) and semantic textual similarity (Gao et al., 2021); 2) the contextual representation of each token in the sentence.

3.3 Pre-Training

Pre-Training Data Different from CodeBERT that needs to be pre-trained with pairs of code snippets and code descriptions, we only use code descriptions to pre-train QudeBERT.

Masked Language Model (MLM) Objective In the inputting sentence, we randomly select a sample of tokens and replace them with a special token $[MASK]$. In the MLM task, the representations of $[MASK]$ tokens from the last hidden layer are fed to a softmax function and MLM objective is a cross-entropy loss on predicting the masked tokens. Following Devlin et al. (2018), we select 15% of inputting tokens for three replacement ways: 1) 80% of selected tokens are replaced with $[MASK]$; 2) 10% of selected tokens are left unchanged; 3) the remaining tokens are replaced with a token randomly selected from the vocabulary.

3.4 Two-Stage Fine-Tuning

CoSQA (Huang et al., 2021) only contains about 20K pairs of queries and code descriptions. Intuitively, it is difficult to fine-tune QudeBERT on CoSQA because it is too small (our experimental results on CoSQA support our conjecturation). We follow *Huang et al.*'s study (Huang et al., 2021) that first utilized CodeSearchNet Python Corpus to fine-tune CodeBERT before fine-tuning it on CoSQA, we design a two-state fine-tuning strategy: we first fine-tune QudeBERT with a matching task of question title and its description[2] on Python_Q (we introduce it in Section 4), a large-scale dataset collected by us. After finishing first-state fine-tuning, we further fine-tune QudeBERT on CoSQA.

The Matching of Question Title and Title Description We formulate the matching of question title and title description as a binary classification task. For each question title q_i and title description d_i, we insert $[CLS]$ in front of the sentence and $[SEP]$ at the end. We input q_i and d_i to QudeBERT and use the representation of $[CLS]$ for the following classification task.

[2] Query2Desc cannot use CSN Python corpus because it only consists of pairs of code snippets and code descriptions, but Query2Desc requires pairs of code descriptions and queries. Since it is difficult to collect millions of pairs of code descriptions and queries, we use a similar task that has large-scale dataset to perform the fine-tuning in first state.

Example 1

Title:	Determine the type of an object?
Body:	Is there a simple way to determine if a variable is a list, dictionary, or something else? I am getting an object back that may be either type and I need to be able to tell the difference.

Example 2

Title:	How do you append to a file?
Body:	How do you append to the file instead of overwriting it? Is there a special function that appends to the file?

Figure 4: Examples of Question Title and Body in StackOverflow. Body denotes the title description.

$$q_c = \text{QudeBERT}(q_i), d_c = \text{QudeBERT}(d_i). \quad (1)$$

We build a simple classification layer to perform $q_i - d_i$ matching through a MLP. We concatenate q_i and d_i and feed it to a feed-forward neural network, to get a fusion embedding:

$$f_{q-d} = \tanh\left(\text{Linear}_1([q_i; d_i])\right). \quad (2)$$

We next put the fusion embedding f_{q-d} into a perceptron classifier with sigmoid fucntion:

$$s^{(q_i,d_i)} = \text{sigmoid}(\text{Linear}_2(f_{q-d})) \quad (3)$$

$s^{(q_i,d_i)}$ can be regarded as the semantic similarity of q_i and d_i.

Finally, we train this binary classification model with binary cross-entropy loss function:

$$\mathcal{L}_b = -[y_i \cdot \log s^{(q_i,d_i)} + (1 - y_i) \log 1 - s^{(q_i,d_i)}], \quad (4)$$

where y_i is label of (q_i, d_i).

Fine-tuning for Code Search The fine-tuning for code search is similar to the fine-tuning of the first state. We only need to change the input to the pairs of code descriptions and queries. Then we initialize the weight of QudeBERT from QudeBERT fine-tuned in the first stage and fine-tine it on the corresponding code search dataset.

4 Experimental Settings

Datasets In our experiments, we keep the balance of positive and negative samples and use the following datasets:

- **CodeSearchNet** It is widely used in the code search task and contains about 6M functions from open-source projects in six different programming languages (Go, Java, JavaScript, PHP, Python, and Ruby). About 2M functions are paired with code descriptions obtained from their documentation, which are utilized to simulate queries. We use code descriptions in the corpus to pre-training models on Query2Desc.

- **Python_Q** We collect pairs of question title and title description from StackOverflow[3] because it is one of the largest online platform for coding questions & answers. Additionally, it also collects and releases posts with specific tags on StackExchange[4]. Therefore, we download the posts with Python tag from it and obtain 1,752,776 python questions. For each python question, we divide it into a question title and its corresponding description, as shown in Figure 4. Generally, the question title is usually the summarization of title description, thus having high semantic consistency with it. Then, we pair each question title with its description and another description randomly selected from other python questions. Next, we label pairs of question title and its corresponding description as positive samples and label other pairs as negative samples. Finally, we get 3,505,552 pairs of python question title and title description, half of which are negative samples. We use this dataset to perform the matching task of question title and title description as the fine-tuning of the first state.

- **CoSQA** It contains more than 20K pairs of queries and code snippets, it is also the biggest real-world dataset for the code search task. It is randomly divided into training, validation, and test sets in the numbe of 19,604:500:500. We use training and validation sets to fine-tune all the models, and use test set to evaluate them.

Baseline Methods We simply choose BERT-base (Devlin et al., 2018), RoBERTa-base (Liu et al., 2019), and CodeBERT (Feng et al., 2020) as the baseline models, to compare their performance

[3] https://stackoverflow.com/

[4] https://data.stackexchange.com/

Search Paradigm	Model	Data	MRR@1	MRR@5	MRR
Query2Code	BERT	CSN+CoSQA	13.80	19.87	22.37
	RoBERTa	CSN+CoSQA	21.60	29.73	32.48
	CodeBERT♣	CSN+CoSQA	51.87	52.28	54.41
Query2Desc	BERT	CoSQA	3.20	6.66	9.17
	BERT	Python_Q+CoSQA	**55.00**	**64.97**	**66.38**
	RoBERTa	CoSQA	0.20	0.25	1.01
	RoBERTa	Python_Q+CoSQA	47.80	56.76	58.60
	CodeBERT	CoSQA	0.00	0.42	1.39
	CodeBERT	Python_Q+CoSQA	53.00	62.94	64.57

Table 1: Models performance on the code search task. CSN denotes CodeSearchNet Python corpus. For models with Query2Code search paradigm, we highlight the highest number among models. For models with Query2Desc search paradigm, we highlight the highest number among models with the same encoder. ♣ : MRR results from Huang et al. (2021) and we re-run their public source code to get other results. Data denotes the dataset used in the fine-tuning phase. On Query2Desc, using CoSQA means that pre-trianed model is not apllied two-state fine-tuning.

Search Paradigm	Model	Data	MRR@1	MRR@5	MRR
Query2Code	CodeBERT + CoCLR♣	CSN+CoSQA	61.38	62.34	64.66
Query2Desc	BERT + CoCLR	CoSQA	69.60	77.83	78.58
	BERT + CoCLR	Python_Q+CoSQA	69.60	78.76	79.58
	RoBERTa + CoCLR	CoSQA	59.00	70.62	71.59
	RoBERTa + CoCLR	Python_Q+CoSQA	73.60	81.13	81.83
	CodeBERT + CoCLR	CoSQA	68.00	77.50	78.24
	CodeBERT + CoCLR	Python_Q+CoSQA	**75.40**	**82.32**	**83.09**

Table 2: Model performance when combining Query2Desc with CoCLR.

on Query2Code and Query2Desc. Besides baseline methods, we also find that CoCLR (Huang et al., 2021), as a contrastive learning method, can improve the performance of CodeBERT. Hence, we also explore whether it can improve model performance on Query2Desc. To apply CoCLR, Huang et al. (2021) built a new training objective:

$$\mathcal{L} = \mathcal{L}_b + \mathcal{L}_{ib} + \mathcal{L}_{qr}, \tag{5}$$

where $\mathcal{L}_b$ is a binary cross-entropy loss function, $\mathcal{L}_{ib}$ is the loss function of sample with in-batch data (for a sample in a batch, the other samples in the batch can be regarded as negative sample):

$$\mathcal{L}_{ib} = -\frac{1}{n-1}\sum_{\substack{j=1 \\ j \neq i}}^{n} \log(1 - s^{(q_i, d_j)}), \tag{6}$$

where n is batch size. $\mathcal{L}_{qr}$ is the loss function of the example with query-written augmentation:

$$\mathcal{L}_{qr} = \mathcal{L}'_b + \mathcal{L}'_{ib}, \tag{7}$$

$\mathcal{L}'_b$ and $\mathcal{L}'_{ib}$ are similar to $\mathcal{L}_b$ and $\mathcal{L}_{ib}$ by only changing q_i to q'_i. The latter is a re-written query by randomly switching the position of two words in query q_i.

Evaluation Metric Following the prior studies, we use Mean Reciprocal Rank (MRR) as the evaluation metric on the code search task. Specially, we calculate MRR of top-1 search result (MRR@1), top-5 search results (MRR@5), and all search results (MRR), respectively.

5 Experimental Results and Analysis

5.1 Effectiveness of Query2Desc

We compare the performance of BERT (Devlin et al., 2018), RoBERTa (Liu et al., 2019), and

Search Paradigm	Model	Data	Top-1	Top-5	Top-10
Query2Code	CodeBERT	CSN+CoSQA	301	379	422
	CodeBERT+CoCLR	CSN+CoSQA	321	398	415
Query2Desc	BERT+CoCLR	Python_Q+CoSQA	348	456	477
	RoBERTa+CoCLR	Python_Q+CoSQA	368	**461**	478
	CodeBERT+CoCLR	Python_Q+CoSQA	**377**	460	**485**

Table 3: Searching results on the test set of CoSQA. Top-k expresses whether the right search result in the Top-k resutls returned by models.

Query	Code descriptions
how to prevent a file from modifying python	Make file user readable if it is not a link
how to cehck if somethign is a constant python	A static value does not change at runtime at compile time
object is not callable range funtion python	Return possible range for min function
python function get all objects of certain type	Get object if child already been read or get child
how to load data from url with python	Recieving the JSON file from uulm
clear an numpy array from python	Free the underlying C array
get largest date from a list python	Given a QuerySet and the name of field containing datetimes return the latest most recent date
python update docstring while inheretance	Set of method to of method in its parent class
how to change to days in python	Converts time strings to integer seconds param time string return integer seconds
how do functions in python know the parametr type	Return true if the string is a mathematical symbol
python get text of response	Turns response into a properly formatted json or text object
remove a value from all keys in a dictionary python	Returns a copy of dct without keys keys
python function compare length of 2 strings	Return the number of characters in two strings that don t exactly match

Table 4: Some queries that the state-of-the-art model on Query2Desc cannot search the right result.

Dataset	Size	Code Avg. Len	Desc Avg. Len	Query Avg. Len
CSN	2,070,536	117.3	17.0	-
CSN-Python	457,461	117.3	16.4	-
Python_Q	1,752,776	-	9.5	214.0
CoSQA	20,604	39.8	11.6	6.6

Table 5: The statistics of datasets we use in the experiments. CSN-Python is the Python corpus in CSN dataset. Code Avg. Len, Desc Avg. Len, and Query Avg. Len are the average length of code snippets, code descriptions, and queries. Especially, for Python_Q dataset, Desc Avg. Len and Query Avg. Len are the average length of Python question title and title description.

CodeBERT (Feng et al., 2020) on Query2Code and Query2Desc. The detailed experimental results can be seen in Table 1. On Query2Code, CodeBERT with two-state fine-tuning on CSN and CoSQA achieve the state-of-the-art result, which shows its effectiveness. On Query2Desc, we find that when we directly fine-tune baseline models on CoSQA, the experimental results are significantly terrible, which supports our conjecture that CoSQA is too small to fine-tune baseline models. We also find that when we apply the two-state fine-tuning strategy to baseline models, they all get significant performance improvement and outperform CodeBERT on Query2Code. From the results, Query2Desc is an effective search paradigm for code search.

Query2Desc with CoCLR As shown in Table 2, CodeBERT on Query2Code can further gets improvement of 9.94% in terms of averaged MRR when applying CoCLR to it, which is the state-of-the-art result on Query2Code. We thus combine Query2Desc with CoCLR, to explore whether CoCLR is also effective on our proposed search paradigm. The experimental results are good and baseline models outperforms the state-of-the-art result on Query2Code by 4.28% to 17.48% in terms of averaged MRR, which shows the universal of Query2Desc. Moreover, we also observe that using CoCLR on Query2Desc enables baseline models to obtain competitive results by directly fine-tuning them on CoSQA. The reason is that contrastive learning is accompanied by data augmentation, which enables us to directly fine-tune baseline models on enlarged CoSQA. To sum up, combining Query2Desc with CoCLR makes baseline models get the state-of-the-art results on code search.

Statistics of Code Search Results Except for calculating MRR scores for models, we also count the search results of models for 500 queries in the test set. As shown in Table 3, CodeBERT with CoCLR on Query2Desc returns the most right results in top-1 and top-10 search results, and RoBERTa with CoCLR on Query2Desc return the most right results in top-5 search results which means at most 97% of queries can get the right result in top-10 search results when performing code search on Query2Desc. The remaining bad search results motivate us to observe the remaining 15 pairs of code descriptions and queries, to find the reason why our models cannot return the right results for them.

We carefully read the 15 pairs of queries and code descriptions and find that most of them are not in direct semantic similarity (Table 4). For instance, by watching the query "*how to prevent a file from modifying python*" and its corresponding code descriptions "*Make file user readable if it is not a link*", it is hard for us to find the slight semantic relation between these two sentences although we are familiar with Python. Considering that similar pairs are less in the dataset, it makes the model hard to learn the effective semantic matching for the above obscure pair of queries and code descriptions. We think that this problem may be caused by the inconsistent viewpoint between users and experienced developers. The former tends to use simple words to express their search purpose and the latter is accustomed to using more professional words to describe the function of code snippets.

5.2 Analysis: Data Size used in Query2Code and Query2Desc

We think that it is necessary to compare the scale of datasets used on Query2Code and Query2Desc. The reason is that if models on Query2Desc are trained with more data and get better results, it is unfair to models on Query2Code. We count the scale of each dataset (Table 5). In the pre-training phase, models on Query2Code are trained with 2,070,536 pairs of code snippets and code descriptions in the CSN dataset. By contrast, models on QueryDesc only need part of code descriptions in the CSN dataset. In the fine-tuning phase, although models on Query2Code and Query2Desc all perform two-stage fine-tuning, Python_Q is a larger dataset than CSN-Python. By comprehensively comparing datasets used on Query2Desc and Query2Code, we think that they use almost equal amounts of data. We thus get our conclusion: Query2Desc is more useful than Query2Code because it eliminates the problem of gap transferring and expansion between code snippets and queries. Besides, Query2Desc enables superior pre-trained models in NLP to be easily transferred to the code search task.

6 Conclusion

In this paper, we focus on the problem of gap transferring and expansion between code snippets and queries. We propose a new search paradigm, Query2Desc, for the code search task, by which we transform the semantic matching of queries and code snippets into the semantic matching of queries and code descriptions. We conduct a series of experiments to demonstrate that models on Query2Desc effectively eliminate the potential gap transferring and expansion in Query2Code. We also provide a specific analysis to show that models on Query2Desc perform badly if code descriptions and queries do not have obvious semantic similarity while existing the obscure semantic relation. In the future, we believe that Query2Desc can be useful for other types of code search task, such as code-to-code search, which refers to description-to-description search in our paradigm.

References

Miltiadis Allamanis, Earl T Barr, Premkumar Devanbu, and Charles Sutton. 2018. A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR)*, 51(4):1–37.

Miltos Allamanis, Daniel Tarlow, Andrew Gordon, and Yi Wei. 2015. Bimodal modelling of source code and natural language. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 2123–2132.

Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. 2018. code2seq: Generating sequences from structured representations of code. *arXiv preprint arXiv:1808.01400*.

Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. code2vec: Learning distributed representations of code. *Proceedings of the 46th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, 3(POPL):1–29.

Sushil Bajracharya, Trung Ngo, Erik Linstead, Yimeng Dou, Paul Rigor, Pierre Baldi, and Cristina Lopes. 2006. Sourcerer: a search engine for open source code supporting structure-based search. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 681–682.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Sen Fang, You-Shuai Tan, Tao Zhang, and Yepang Liu. 2021. Self-attention networks for code search. *Information and Software Technology*, 134:106542.

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. *arXiv preprint arXiv:2002.08155*.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. Simcse: Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv:2104.08821*.

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

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. 2020. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366*.

Rajarshi Haldar, Lingfei Wu, Jinjun Xiong, and Julia Hockenmaier. 2020. A multi-perspective architecture for semantic code search. *arXiv preprint arXiv:2005.06980*.

Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep code comment generation. In *Proceedings of the 2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, pages 200–20010. IEEE.

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. *arXiv preprint arXiv:2105.13239*.

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*.

Kisub Kim, Dongsun Kim, Tegawendé F Bissyandé, Eunjong Choi, Li Li, Jacques Klein, and Yves Le Traon. 2018. Facoy: a code-to-code search engine. In *Proceedings of the 40th International Conference on Software Engineering*, pages 946–957.

Chao Liu, Xin Xia, David Lo, Cuiyun Gao, Xiaohu Yang, and John Grundy. 2022a. Opportunities and challenges in code search tools. *ACM Computing Surveys*, 54(9):1–40.

Chao Liu, Xin Xia, David Lo, Zhiwe Liu, Ahmed E. Hassan, and Shanping Li. 2022b. Codematcher: Searching code based on sequential semantics of important query words. *ACM Transactions on Software Engineering and Methodology*, 31(1):1–37.

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

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Meili Lu, Xiaobing Sun, Shaowei Wang, David Lo, and Yucong Duan. 2015. Query expansion via wordnet for effective code search. In *Proceedings of the 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 545–549. IEEE.

Fei Lv, Hongyu Zhang, Jian-guang Lou, Shaowei Wang, Dongmei Zhang, and Jianjun Zhao. 2015. Codehow: Effective code search based on api understanding and extended boolean model (e). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 260–270. IEEE.

Usman Naseem, Imran Razzak, Katarzyna Musial, and Muhammad Imran. 2020. Transformer based deep intelligent contextual embedding for twitter sentiment analysis. *Future Generation Computer Systems*, 113:58–69.

Liming Nie, He Jiang, Zhilei Ren, Zeyi Sun, and Xiaochen Li. 2016. Query expansion based on crowd knowledge for code search. *IEEE Transactions on Services Computing*, 9(5):771–783.

Mohammad M Rahman, Chanchal K Roy, and David Lo. 2019. Automatic query reformulation for code search using crowdsourced knowledge. *Empirical Software Engineering*, 24(4):1869–1924.

Mohammad Masudur Rahman. 2019. Supporting code search with context-aware, analytics-driven, effective query reformulation. In *Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 226–229. IEEE.

Vighnesh Shiv and Chris Quirk. 2019. Novel positional encodings to enable tree-based transformers. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Jianhang Shuai, Ling Xu, Chao Liu, Meng Yan, Xin Xia, and Yan Lei. 2020. Improving code search with co-attentive representation learning. In *Proceedings of the 28th International Conference on Program Comprehension*, pages 196–207.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Yao Wan, Jingdong Shu, Yulei Sui, Guandong Xu, Zhou Zhao, Jian Wu, and Philip S Yu. 2019. Multi-modal attention network learning for semantic source code retrieval. *arXiv preprint arXiv:1909.13516*.

Thomas Wolf, Julien Chaumond, Lysandre Debut, Victor Sanh, Clement Delangue, Anthony Moi, Pierric Cistac, Morgan Funtowicz, Joe Davison, Sam Shleifer, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45.

Shufan Zhou, Beijun Shen, and Hao Zhong. 2019. Lancer: Your code tell me what you need. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1202–1205. IEEE.

A Implementation Details

We initialize all baseline models with their corresponding pre-trained models. For BERT and RoBERTa, we intialize them with *bert-base-uncased*[5] and *roberta-base*[6]. For CodeBERT, we intialize it with *microsoft/codebert-base*[7]. We use `transformers` (Wolf et al., 2020) package to perform all the experiments on an NVIDIA Tesla V100 GPU with 32GB memory. We set batch size to 256 and use the AdamW (Loshchilov and Hutter, 2017) optimizer with learning rate 1e-5. We train each model for 10 epochs and evaluate it every epoch on the validation set of CoSQA (Huang et al., 2021). We keep the best epoch for the final evaluation on the test set.

B Testing Details

To effectively evaluate the performance of models, we collect all positive pairs in CoSQA and build a codebase with 6,267 different pairs of code descriptions and code snippets. For models on Query2Code, we directly search code snippets according to the given query. For models on Query2Desc, we search code descriptions according to the given query.

[5]https://huggingface.co/bert-base-uncased
[6]https://huggingface.co/roberta-base
[7]https://huggingface.co/microsoft/codebert-base