Repo4QA
Answering Coding Questions via Dense Retrieval on GitHub Repositories
Chen, Minyu; Li, Guoqiang; Ma, Chen; Li, Jingyang; Fu, Hongfei

Published in:
Proceedings of the 29th International Conference on Computational Linguistic

Published: 01/10/2022

Document Version:
Final Published version, also known as Publisher's PDF, Publisher's Final version or Version of Record

License:
CC BY

Publication record in CityU Scholars:
Go to record

Publication details:

Citing this paper
Please note that where the full-text provided on CityU Scholars is the Post-print version (also known as Accepted Author Manuscript, Peer-reviewed or Author Final version), it may differ from the Final Published version. When citing, ensure that you check and use the publisher's definitive version for pagination and other details.

General rights
Copyright for the publications made accessible via the CityU Scholars portal is retained by the author(s) and/or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights. Users may not further distribute the material or use it for any profit-making activity or commercial gain.

Publisher permission
Permission for previously published items are in accordance with publisher's copyright policies sourced from the SHERPA RoMEO database. Links to full text versions (either Published or Post-print) are only available if corresponding publishers allow open access.

Take down policy
Contact lbscholars@cityu.edu.hk if you believe that this document breaches copyright and provide us with details. We will remove access to the work immediately and investigate your claim.
Repo4QA: Answering Coding Questions via Dense Retrieval on GitHub Repositories

Minyu Chen¹, Guoqiang Li¹∗, Chen Ma², Jingyang Li¹, Hongfei Fu¹
¹Shanghai Jiao Tong University, Shanghai, China
²City University of Hong Kong, Hong Kong S.A.R.
{minkow, li.g, lijjjjj}@sjtu.edu.cn
chenma@cityu.edu.hk, fuhf@cs.sjtu.edu.cn

Abstract

Open-source platforms such as GitHub and Stack Overflow both play significant roles in current software ecosystems. It is crucial but time-consuming for developers to raise programming questions in coding forums such as Stack Overflow and be navigated to actual solutions on GitHub repositories. In this paper, we dedicate to accelerating this activity. We find that traditional information retrieval-based methods fail to handle the long and complex questions in coding forums, and thus cannot find suitable coding repositories. To effectively and efficiently bridge the semantic gap between repositories and real-world coding questions, we introduce a specialized dataset named Repo4QA, which includes over 12,000 question-repository pairs constructed from Stack Overflow and GitHub. Furthermore, we propose QuRep, a CodeBERT-based model that jointly learns the representation of both questions and repositories. Experimental results demonstrate that our model simultaneously captures the semantic features in both questions and repositories through supervised contrastive loss and hard negative sampling. We report that our approach outperforms existing state-of-art methods by 3%-8% on MRR and 5%-8% on P@1.

1 Introduction

With the increasing popularity of software development, Stack Overflow and GitHub (two large-scale communities widely recognized in open source ecosystems) have attracted growing research interests. As the idiom goes, “Don’t reinvent the wheel”. Tackling programming problems with existing codes and documents is more effective and economical, since various resources in repositories can provide more helpful information than text-formed answers. Specifically, developers can get help and advice to solve the technical challenges they face, and be provided a variety of solutions and tools in repositories on GitHub. As a vast number of challenges have already been considered and settled by the community, sophisticated schemes posted on GitHub can help satisfy requirements or solve problems discussed in Stack Overflow. Naturally, many answers in Stack Overflow provide links to GitHub repositories, and a large amount of these answers are acknowledged to be high-quality and helpful by the community. This phenomenon is worth investigating to determine its contribution to the efficiency improvement and the code reuse of the open source ecosystem.

Motivated by the interplay between Stack Overflow and GitHub, we introduce a novel question-repository matching task. Given a natural-language-formed question in the programming domain, we aim to search for the most relevant GitHub repository from all candidate repositories as the answer. Figure 1 illustrates the interaction between a question and a repository, where the key information for problem-solving is framed.

To this end, we introduce Repo4QA, a dataset consisting of 12,995 question-repository pairs² for complex coding question solving. The questions are collected from Stack Overflow, and the repositories are crawled from GitHub through the hyperlinks provided in corresponding Stack Overflow answers. Each repository is instrumental for troubleshooting confirmed by forum users with upvotes. Such that proper repositories can be used as answers to coding questions.

Different from previous information retrieval (IR) tasks, the challenge of the proposed task lies in the long-form questions. Typical IR methods and tricks, such as query expansion, are designed for short and keyword-based user inputs, while our task focuses on questions written in natural lan-

¹Corresponding author.
²Our dataset and code are available at https://github.com/Minkow/Repo4QA
Different from code searching task (Husain et al., 2019; Cambronero et al., 2019), our task is supposed to find a reasonable semantic alignment between two long-form sentences. Questions and repositories have more complex structures and richer information than short-form web queries and code snippets. Compared with community-based QA task (Qiu and Huang, 2015a; Zhao et al., 2017), our task is a cross-platform task resulting in semantic gaps between questions and answers, which is more challenging for traditional IR-based methods such as BM-25 (Robertson et al., 1995). Besides, unlike common questions, questions about programming are more challenging and specialized. The terminology of programming is widely used to form questions; there are even some questions described with codes. Moreover, complex models considering more interactions between QA pairs are more computationally expensive in our task. Finally, the interactions inside queries or documents corpus should not be neglected. The relationship of the elements in the same semantic category is informative while discriminating positive samples from negatives.

To address the aforementioned issues, we propose a novel jointly learning scheme, QuRep, for Question and Repository pairs. QuRep employs CodeBERT (Feng et al., 2020) as the initial weight of its transformer-based encoder. Our work leverages the supervised contrastive loss for better separating elements from each other in the representation space. An unsupervised hard negative sampling strategy enables better discrimination performance. The shared weights between encoders reduce the computation cost, while the relevance between questions and answers can be ranked by the similarity score of embedding vectors. With experimental evaluation and comprehensive analyses, we show that our method strongly outperforms baselines.

To summarize, the main contributions of this paper are concluded into three points respectively.

- **Dataset**: A novel cross-platform Question-Answering task is presented, aiming at answering real-world programming questions with existing GitHub repositories. We especially collect a dataset, Repo4QA, for this task.

- **Methodology**: A practical joint embedding model, QuRep, is proposed to optimize the classification and contrastive loss, which can represent both questions in language and repositories.

- **Experiment**: Experimental results demonstrate the effectiveness and efficiency of the proposed QuRep model compared with baselines.

## 2 Related Work

### Datasets.** Existing datasets in the programming domain focus on text-code interaction. CodeSearchNet (Husain et al., 2019), Deep Code Search (Gu et al., 2018) and CoDesc (Hasan et al., 2021) collect large-scale corpus of code snippet with corresponding descriptions. CoSQA (Huang et al., 2021) collects pairs of web query and function code for code question answering. Stack Overflow resources are mined (Yin et al., 2018) as long-form natural language queries to retrieval code snippets (Nie et al., 2016; Yao et al., 2018; Heyman and Van Cutsem, 2020). CodeXGLUE (Lu et al., 2021) includes text-to-code generation task and code memorization task. The only text-to-text task

![Figure 1: An example of interaction between question at Stack Overflow and repository at GitHub.](image)
is documentation translation in CodeXGLUE. Besides, LinkSO (Liu et al., 2018) discovered the similarity between repositories.

**Neural Matching Networks.** Ranking methods are widely used in text matching and semantic search tasks, such as community-based question answering (Qiu and Huang, 2015b; Zhang et al., 2021b), retrieval-based question answering (Qu et al., 2020; Cohen et al., 2018), and visual question answering (Lee et al., 2020). Specifically, in the programming domain, traditional IR-based ranking models regard code as text and match keywords in queries (Bajracharya et al., 2006). Recently, deep learning based methods represent coding questions and answers with vectors and leverage similarities to rank answers (Gu et al., 2018; Cambronero et al., 2019; Wan et al., 2019).

Considering the computational cost for matching, representation-based learning approaches (Huang et al., 2013) encode a query paired with a document separately into a vector and judge the relevancy by the similarity of vectors. Towards a better representation of repositories, paper2repo (Shao et al., 2020) maps the embeddings of academic papers and repositories into the same space for ranking and recommendation. Very recently, pre-trained models, including CodeBERT (Feng et al., 2020) that trained from data in the programming domain, have been applied to improve representation learning.

**LM Based Retrieval and Ranking.** Pretrained language models (Kenton and Toutanova, 2019; Liu et al., 2019b) dramatically advance the state of the art on various NLP tasks. However, limitations on text length and the trade-off of effectiveness and efficiency are important issues, as the cross-attention operations are too expensive in pair-wise cross-encoders. Recent work (Karpukhin et al., 2020; Xiong et al., 2020; Khattab and Zaharia, 2020; Nie et al., 2020; Gao et al., 2021a) try to reduce the computational interaction between query and document and move it to the online re-rank procedure. By storing the representation of documents offline, these methods facilitate cheap runtime costs while achieving promising results on retrieval tasks.

Moreover, contrastive learning on pre-training models has recently been broadly applied to several sentence-level tasks. SBERT (Reimers and Gurevych, 2019) uses siamese and triplet network to derive embedding of two sentences, then fine-tune the model to yield useful sentence embeddings. Some work aims to improve BERT sentence embeddings in an unsupervised way (Gao et al., 2021b; Kim et al., 2021) by data augmentation.

### Dataset Description

**3 Dataset Description**

**3.1 Repo4QA Dataset**

**Questions.** We collect complex programming questions from the well-known coding forum Stack Overflow. Stack Overflow provides data dump 3 from 2014 to 2021. To avoid ambiguous answers such as “We can deploy A after B to tackle this”, answers less than 200 characters that contain only a hyperlink to a GitHub repository are selected. Responders are required not only to get through the requirements raised by questioners, but also to be familiar with repositories stored on GitHub. The goal of our research is to fill in the gap between the questioner and various open source tools.

To filter out answers without specific repository-for-solution and repository-structured intent such as bug-reporting discussion, we discard answers linking to particular resources in repositories such as issues, commits, and releases (e.g., the question is “Assets serving paths in Rails 3.1 - how to customize it?” and the answer is “GitHub.com/chriseppstein/compass/issues/337 - there is a big discussion about it”). We control the quality and correctness of answers by only mining posts with at least one upvote. After removing answers with inaccessible GitHub repository links, these answers and corresponding questions compose 12,995 QA pairs with 12,713 unique questions. Codes are marked with [code] token to help our model learn the combination of natural language and code in questions.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Type</th>
<th>Samples</th>
<th>Avg. length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large</td>
<td>Question</td>
<td>12,995</td>
<td>7.97 + 104.50</td>
</tr>
<tr>
<td></td>
<td>Repository</td>
<td>9,954</td>
<td>9.79 + 572.58</td>
</tr>
<tr>
<td>Small</td>
<td>Question</td>
<td>3,766</td>
<td>8.02 + 105.73</td>
</tr>
<tr>
<td></td>
<td>Repository</td>
<td>2,862</td>
<td>9.77 + 688.03</td>
</tr>
</tbody>
</table>

Table 1: The statistics of Repo4QA dataset. Avg. length means title length + body length for question, and description length + Readme length for repository. Readme file has the maximum character length of 8192.

3https://archive.org/details/stackexchange
Repositories. We crawl repositories through the GitHub API with given hyperlinks in answers. For our task, we collect basic information such as the name, description, topics (also called tags), and stars of a repository. The description is short textual documentation to describe a repository briefly. Topics are keywords to classify a repository. For instance, the tag “python” indicates the programming language that the repository uses, and the tag “deep-learning” shows that the repository is used in the deep learning domain. In addition, the Readme file is also obtained for detailed documentation. Most repositories introduce the main contribution and usage in the head of the Readme file. We investigate 50 repositories randomly and find that the most informative part of a Readme file is the starting 2 or 3 paragraphs, which is far less than 512 tokens. Hence, we cut the first 8192 characters of a Readme file after text cleaning to represent the repository in natural language. Likewise, 9,663 unique repositories are investigated in this step, in which 2,862 repositories have at least one topic.

Construction. Questions and repositories are aligned according to QA pairs mined in questions to constitute a QA pair sample. For each pair, the original answer is replaced by the repository mentioned. All samples in the small dataset have at least one topic, while most repositories in the large dataset lack topics. The statistics of both datasets are listed in Table 1.

Comparison. To the best of our knowledge, this is the first dataset applied to solving complex realistic programming problems with existing web resources. In this part, we conduct a comparison between Repo4QA and datasets from two aspects: 1) Code intelligence and 2) Community-based QA. Datasets in the programming domain (Yao et al., 2018; Nie et al., 2016; Yin et al., 2018; Heyman and Van Cutsem, 2020) tend to adopt only the title, content, and tags, while each repository served as an answer candidate has descriptions and documentation. Note that tags may not be contained in some repositories. Given a question in a natural language question set \( q \), and a set of repositories \( R = \{ r_1, r_2, \ldots, r_n \} \) from GitHub, our main task is to find the most helpful repository \( r \in R \) to solve the question \( q \).

Due to the limitation of computation resources in real-world applications, joint embedding is an effective and efficient way to find repositories related to the raised question. Ideally, we train a model that jointly learns the embedding of \( Q \) and \( R \) with a transformer-based encoder network. Specifically, given any question and its dense representation \( q_i \) and repositories’ embedding \( r_i^+ \), \( r_i^- \), where \( r_i^+ \) is one of the answer to \( q_i \), and \( r_i^- \) is not related to \( q_i \). We expect the model to distinguish \( r_i^+ \) from others via similarity metrics, that is, to make \( s(q_i, r_i^+) \) and \( s(q_i, r_i^-) \) satisfying the inequality: \( s(q_i, r_i^+) > s(q, r_i^-) \), where \( s \) denotes a similarity function, e.g., cosine similarity or Euclidean distance-based similarity. Then we rank all the answers for a given question according to the similarity between them.

4 Methodology

4.1 Task Description

Before diving into the details of our model, we first describe some notations used in the answer selection problem. Each question consists of a title, content, and tags, while each repository served as an answer candidate has descriptions and documentation. Note that tags may not be contained in some repositories. Given a question in a natural language question set \( q \), and a set of repositories \( R = \{ r_1, r_2, \ldots, r_n \} \) from GitHub, our main task is to find the most helpful repository \( r \in R \) to solve the question \( q \).

We present our QuRep in Figure 2. Different from the natural language in common domains, the language used in the programming domain contains various out-of-vocabulary words, e.g., “flask” is a tool used for web in Python programming. To solve this problem, we leverage CodeBERT as our text encoder. CodeBERT is a bi-modal pre-trained RoBERTa-based (Liu et al., 2019b) model for natural language (NL) and programming language.
Figure 2: The QuRep applies a weight-sharing CodeBERT for encoding questions and repositories. Similarity is computed between model outputs and embeddings stored in Cross-Batch Memory for model training.

(PL) tasks. It is a bidirectional Transformer with 12 layers, 768 dimensional hidden states and 12 attention heads pre-trained on the large-scale CodeSearchNet (Husain et al., 2019) corpus. CodeBERT achieves the state-of-the-art in most NL-PL, tasks such as the natural language code search and code documentation generation. By expanding the vocabulary of RoBERTa, CodeBERT can properly represent programming terms that occur in the training corpus, especially for word-combining terminologies commonly used in programming. For example, “pyflask” is an OOV word in most models’ vocabulary, but the WordPiece encoding will split “pyflask” into “py” and “flask”.

For each question with title, body, and tags, we put a [CLS] token at the beginning of these three parts and separate them by [SEP] after tokenization. Then we feed the tokenized sequences into CodeBERT to acquire pooled contextualized representations of them, respectively. A [Q] is placed in the start to identify the question. In practice, we adopt the mean pooling value of contextual representations as the output of CodeBERT. Similarly, for each repository, we conduct the same manner as its description, Readme documentation, and topics. The weights are shared between the model for both question encoding and answer encoding.

As few Readme files exceed the token length limit of 512 in CodeBERT, we only take the first 509 tokens (3 tokens are left for [Q]/[A], [CLS] and [SEP]) of the Readme file as the documentation. In practice, the head content of Readme file is descriptive and summative enough, which is more informative than the usage description and changelogs in the later part of Readme file.

4.3 Loss Functions

Our approach considers three kinds of losses, namely discrimination loss, supervised contrastive loss, and classification loss. These loss functions are designed for specific tasks to better represent questions and repositories.

Discrimination Loss. Given a randomly sampled minibatch of positive pairs \( D = (q_i, r_i)^N \), the goal of Discrimination loss is to separate the answer \( r_i \) apart from all other \( 2N - 2 \) repositories in the same batch \( D \). It matches the target to satisfy the inequality:

\[
\text{sim}\left(q^+, r^+\right) > \text{sim}\left(q^+, r^-\right),
\]

where \( \text{sim}(\cdot) \) represents the similarity. We choose cosine similarity as the metric. Following previous work (Karpukhin et al., 2020), the discrimination loss is defined as the negative log likelihood of positive pairs of instances:

\[
\mathcal{L}_D = -\log \frac{e^{s(q_i, r_i)/\tau}}{e^{s(q_i, r_i)/\tau} + D},
\]

where \( \tau \) refers to the temperature parameter, \( D = \sum_{j \neq i} e^{s(q_i, r_j)/\tau} \) denotes the normalization part.

Supervised Contrastive Loss. The discrimination loss discussed above focuses on modeling the distance between questions and answers. However, compared with other QA or NLI tasks (Ren et al., 2021; Zhang et al., 2021a), our questions are more complex and long-formed, which means different questions implicate diverse semantic information. We insist that a good dense representation model should not only control the distance of questions and answers, but can also represent the semantic difference between questions and repositories. Hence, we further leverage the supervised contrastive loss, to enforce the model learn to distinguish similar questions and answers, that is,

\[
\text{sim}(q^+, r^+), \text{sim}(q^+, r^-), \text{sim}(q^+, r^+) > \text{sim}(r^+, r^-).
\]

We also employ the negative log likelihood to achieve the supervised contrastive loss at the instance level:

\[
\begin{align*}
\mathcal{L}_{Cq} &= -\log \frac{e^{s(q_i, r_i)/\tau}}{e^{s(q_i, r_i)/\tau} + Cq}, \\
\mathcal{L}_{Cr} &= -\log \frac{e^{s(q_i, r_i)/\tau}}{e^{s(q_i, r_i)/\tau} + Cr},
\end{align*}
\]
where $Cq = \sum_{j \neq i} e^{s(q, q_j)/\tau}$, and $Cr = \sum_{j \neq i} e^{s(r, r_j)/\tau}$. They refer to the similarity of questions and repositories themselves, respectively.

We observe that the difference between $L_{Cq}$, $L_{Cr}$, and $L_D$ is the normalization part. Instead of simply summing them up, we can uniformly conclude these different types of loss functions into one negative log likelihood as the following Discrimination-Contrastive loss:

$$L_{DC} = -\log \frac{e^{s(q, r_i)/\tau}}{e^{s(q, r_i)/\tau} + D + \alpha \cdot Cq + \beta \cdot Cr},$$

where the weight hyper-parameter $\alpha$ and $\beta$ control the importance of distinguishing similar questions and repositories. The combined Discrimination-Contrastive loss consists of four interactions:

1. $q^+ \leftrightarrow r^+$: The main element that gathers paired question and repository together.
2. $q^+ \leftrightarrow q^-$: The factor that separates the embedding of questions.
3. $r^+ \leftrightarrow r^-$: The component that makes repositories to be distant from each other.
4. $q^+ \leftrightarrow r^-$: An important role that cause unmatched question-repository pairs segregated.

**Hard Negative for Classification Loss.** Previous studies (Luan et al., 2021; Karpukhin et al., 2020; Xiong et al., 2020) provide effective methods of hard negative sampling, and further demonstrate the importance of hard negatives. Ideally, we aim to pay more attention to these hard negatives, which are not semantically similar but are mapped close to the anchor in the vector space.

We employ BM25 as the base sentence matching algorithm. For each question in the training set, we query it in the repository corpus, and then select the top 10 results as the hard negatives except the correct answer. Then we construct triplets $(q^+, r^+, r^-)$ as training instances, where the negative $r^-$ is randomly sampled from these hard negatives at a ratio of $p$ to avoid the model collapse. We minimize the cross entropy loss of enhanced representations of $q$ and $r$ as follows:

$$L_{CE} = CrossEntropy(f(q, r, |q - r|), y),$$

where $f$ denotes a feed-forward classification network, and $y$ represents the classification label.

**Overall Loss.** The overall loss is the weighted sum of the DC loss and the CE loss, which is:

$$L = L_{DC} + \gamma L_{CE},$$

where $\gamma$ is a hyper-parameter to balance the loss.

### 4.4 Cross-Batch Memory Augmentation

Cross-batch memory (XBM) (Wang et al., 2020) can considerably boost the performance of contrastive learning tasks. The XBM module stores embeddings and labels for data points. It is maintained as a first-input-first-output (FIFO) queue. Enqueue and dequeue procedures are conducted when a mini-batch arrives. As mentioned above, for an anchor question $q_i$, we pair it with rest $M - 1$ repositories $\{r_j | j \neq i\}$ in the M-size mini-batch as negative pairs. In practice, those heavyweight BERT-based model has acute GPU memory cost issues. The size of mini-batch is often limited in NLP tasks using BERT-based models. By pairing anchors with samples stored in XBM, the information provided by negative pairs is significantly enriched.

### 5 Experiments

#### 5.1 Experimental Setup

**Dataset.** We conduct experiments on the Repo4QA-small dataset, by randomly splitting Repo4QA-small dataset into 2,966/400/400 for training/testing/validation. For repositories retrieval, we evaluate the performance from 3 different corpora: the test split, the whole Repo4QA-small repositories, and the whole Repo4QA-large repositories, which is a more realistic setting since the repositories do not occur during the training period. We always only raise all 400 questions in the testing set. However, we retrieve answers from our dataset with different scales. As for the test split, we search for the best answer in the 400 repositories. For the Repo4QA-small split, the range of ranking is the 400 repositories in the testset and the rest 3,366 repositories in the Repo4QA-small set. Note that though some repositories have occurred in the training step, we never learn the relationship between questions and these repositories. For the Repo4QA-large split, we seek suitable repositories among the union of the 400 repositories in the testset and all the repositories in the Repo4QA-large set. If given more repositories, such as whole GitHub repositories, a practical solution is to filter
Table 2: Experimental results on the Repo4QA-small test set. The best figure of MRR and P@1 metric is in bold. Our QuRep model outperforms both unsupervised and supervised baselines.

<table>
<thead>
<tr>
<th>Models</th>
<th>Test</th>
<th>Small</th>
<th>Large</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MRR</td>
<td>P@1</td>
<td>P@5</td>
</tr>
<tr>
<td>BERT-base</td>
<td>7.98</td>
<td>4.25</td>
<td>9.50</td>
</tr>
<tr>
<td>CodeBERT-base</td>
<td>2.22</td>
<td>0.25</td>
<td>2.25</td>
</tr>
<tr>
<td>Glove</td>
<td>19.82</td>
<td>13.00</td>
<td>26.00</td>
</tr>
<tr>
<td>BM25</td>
<td>43.22</td>
<td>35.25</td>
<td>51.75</td>
</tr>
<tr>
<td>Universal Sentence Encoder</td>
<td>62.72</td>
<td>49.75</td>
<td>78.25</td>
</tr>
<tr>
<td>S-BERT</td>
<td>80.23</td>
<td>72.00</td>
<td>91.75</td>
</tr>
<tr>
<td>Siamese-CodeBERT</td>
<td>79.37</td>
<td>70.25</td>
<td>88.50</td>
</tr>
<tr>
<td>Triplet-CodeBERT</td>
<td>82.96</td>
<td>76.00</td>
<td>89.75</td>
</tr>
<tr>
<td>QuRep-BERT-base</td>
<td>77.23</td>
<td>70.25</td>
<td>87.00</td>
</tr>
<tr>
<td>QuRep (ours)</td>
<td><strong>86.26</strong></td>
<td><strong>81.00</strong></td>
<td><strong>93.00</strong></td>
</tr>
</tbody>
</table>

**Metrics.** We adopt two common metrics widely used in answer ranking to measure the effectiveness of our proposed model, namely, Mean Reciprocal Rank (MRR) and Precision@K, which means the rate that correct answer appears in the top-K ranked candidates. In practice, we evaluate the performance of $K = 1$ and $K = 5$.

**Baselines.** As Repo4QA is a new challenge, no model is specifically designed for it. Existing methods such as ColBert (Khattab and Zaharia, 2020) focus on passage ranking tasks such as MS MARCO (Nguyen et al., 2016), with short queries and related passages. While query expansion methods (Nogueira et al., 2019b,a) are not so helpful because of the complexity of our questions. Pairwise cross-encoders are more suitable for the re-rank task after we retrieve the dense representation for the consideration of the effective-efficient trade-off. For a fair comparison, diversified methods for similar tasks, such as sentence embedding and metric learning, are introduced as baselines. We use the exact same processed data for our model as the input of these baselines. To compare with our designed loss functions, we train the CodeBERT model with vanilla siamese loss and triplet loss. Appendix B provides implementation details of baselines.

**Implementation Details.** We implement our model based on the Hugging Face transformers library. The initial weight is adopt from microsoft/codebert-base4. We use RAdam (Liu et al., 2019a) as the optimizer, with a cosine annealing learning rate from 1e-5 to 1e-7. The temperature $\tau$ in the loss function is 0.01. Factors of weights are set as $\alpha = \beta = 1$ and $\gamma = 0$. The negative sampling ratio is 0.2. For cross-batch memory, we set the size of the memory bank to 64, and this mechanism starts working from epoch 6.

### 5.2 Model Comparisons

The performance of different approaches on the Repo4QA task is in Table 2. From these experimental results, we have the following observations:

First, our proposed model outperforms all competitive baselines on MRR, P@1, and P@5 metrics. Especially, for the retrieval task on Repo4QA-large dataset, traditional IR-based and word embedding approaches cannot differentiate target from similar repositories, while contrastive-learning methods strongly outperform others. This result demonstrates the difficulty of understanding long-form questions compared with short queries, and the necessity of precise retrieval on large corpora via semantic search instead of lexical matching.

Second, poor performance is shown by BERT

4https://github.com/microsoft/CodeBERT
Table 3: Results with different comparable loss function.

<table>
<thead>
<tr>
<th>Loss</th>
<th>MRR</th>
<th>P@1</th>
<th>P@5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>86.26</td>
<td>81.00</td>
<td>93.00</td>
</tr>
<tr>
<td>Vanilla triplet loss</td>
<td>82.96</td>
<td>76.00</td>
<td>89.75</td>
</tr>
<tr>
<td>Hinge loss</td>
<td>83.72</td>
<td>77.25</td>
<td>91.75</td>
</tr>
<tr>
<td>Circle loss</td>
<td>82.44</td>
<td>75.00</td>
<td>91.25</td>
</tr>
<tr>
<td>InfoNCE</td>
<td>84.44</td>
<td>78.25</td>
<td>92.25</td>
</tr>
</tbody>
</table>

Table 4: Results of ablation study conducted on Test split about model structure.

<table>
<thead>
<tr>
<th>Models</th>
<th>MRR</th>
<th>P@1</th>
<th>P@5</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuRep</td>
<td>86.26</td>
<td>81.00</td>
<td>93.00</td>
</tr>
<tr>
<td>w/o XBM</td>
<td>-3.28</td>
<td>-4.00</td>
<td>-2.50</td>
</tr>
<tr>
<td>w/o CodeBERT</td>
<td>-9.03</td>
<td>-10.75</td>
<td>-6.00</td>
</tr>
<tr>
<td>w/o Constrative Loss</td>
<td>-21.24</td>
<td>-26.75</td>
<td>-13.25</td>
</tr>
<tr>
<td>w/o CE Loss</td>
<td>-0.83</td>
<td>-1.50</td>
<td>-1.00</td>
</tr>
<tr>
<td>w/o Hard Negatives</td>
<td>-1.43</td>
<td>-2.75</td>
<td>-1.25</td>
</tr>
</tbody>
</table>

and CodeBERT if directly employing mean pooling output to similarity computation. The results are even worse than using average GloVe embeddings. Universal Sentence Encoder shows effectiveness among unsupervised learning methods, and SBERT achieves great performance since they are trained on large QA corpora.

Third, nearly all supervised methods achieve higher performance than unsupervised models, though no early interaction, such as cross-attention between questions and repositories, is considered. This phenomenon demonstrates the sound effect of contrastive learning.

Fourth, although some repositories in the Repo4QA-small dataset have been modeled in the training period, the performance still drops a lot compared with the result in the Repo4QA-test. The root cause is the diversity of questions. As the questions are always from the testset, they keep unseen for the model in the inference step. There is also no issue of data leakage in our Repo4QA-small dataset because the training data consists of QA pairs, but the questions in testset never pair with answers in the Repo4QA-small dataset. When questions have a longer form instead of simple keywords query, the semantic information contained is richer and more diversified, which increases the difficulty of encoding.

5.3 Loss Comparisons

We agree that an empirical study for comparable loss functions contributes to the soundness of our method. As reported in Table 3, experimental results demonstrate the hybrid loss we design significantly improves the performance compared with the classical loss functions of contrastive study (Sun et al., 2020; Gao et al., 2021b).

The strongest baseline, InfoNCE, has a similar form to our Discrimination-Contrastive loss. The difference is that we distinguish the effect from questions and repositories as stated in Sect. 4.3.

5.4 Ablations Study

As the experimental results show the same trend in all three splits, in this part, we only conduct experiments in the test split. We consider the model components and loss function components as two main aspects of our ablation studies.

Effects of model components. Removing or replacing components of QuRep decreases the performance, as Table 4 shows. To be more specific, we replace CodeBERT with BERT-base, just as the result of QuRep-BERT-base shows. The performance drop proves the effectiveness of pre-training in the programming domain. Cross-batch memory is proved to be a necessary mechanism for constructing more QA pairs to be trained.

Effects of loss components. We also investigate the ablation of constituents in our proposed hybrid loss function. Each part of our loss contributes to the improvement of performance. As illustrated in Table 4, contrastive loss is the backbone of our learning target, and the hard negative sampling strategy also plays an important role.

5.5 Case Study

To better understand the difficulty of the task and the effectiveness of our solution, we present a case study in this part. As a typical method in the IR field, BM25 is selected as the baseline method to compare. All these cases can be viewed on StackOverflow. We only display key parts of questions/repositories considering the full length.

The first and second cases in Table 5 are examples that QuRep outperforms BM25. Question 1 means to find a dictionary implementation in JavaScript. BM25 returns results containing the keyword “dictionary,” as the repository “stig/json-framework” (now redirected to SBJson/SBJson) mentions NSMmutableDictionary type in Objective-C.
C, which is not so close to the question. In comparison, our proposed model recommends the collection of data structures written in JavaScript. Though the word “dictionary” is not discussed in the repository so frequently, “dictionary” is contained by the concept “data structures.” This is what language models bring us. Case 2 and 3 also show that BM25 tend to return similar repositories at the word level, which results in repositories from different disciplines or programming languages. While QuRep can identify the purpose of questions and recommend better answers.

Case 4 is a bad case in which both models fail. The best answer posted on StackOverflow is “typicode/husk.” QuRep ranks “lerna/lerna” as the best because it discusses both Angular and Git. However, the phrase “unit test” is ignored.

From the case study section, we can learn that understanding the intent of a question is a key factor for our task, especially when the question contains rich semantic information. The relations of concepts are also an important issue to study.

6 Conclusion

In this paper, we introduce an automatically collected novel dataset, Repo4QA, for the proposed task. Furthermore, we propose QuRep, a contrastive learning method to fine-tune CodeBERT for our task. Experimental results demonstrate that our method outperforms baseline models in effectiveness and efficiency. Detailed analyses are conducted to investigate the impact on performance brought by components of our model. We look forward to other applications and more research interest in our task. Moving forward, we plan to deploy our dataset and method to solve programming questions in software engineering practice and consider code stored in repositories for better modeling of multi-modal GitHub repositories.

Acknowledgements

This work is supported by the National Science Foundation of China Grant No. 62161146001.

References


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


Liuyan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. 2019a. On the variance of the adaptive learning rate and beyond. In International Conference on Learning Representations.


Xipeng Qiu and Xuanjing Huang. 2015b. Convolutional neural tensor network architecture for community-based question answering. In IJCAI.


A Dataset comparison

As shown in Table 6, we compare our dataset with several datasets on Code Intelligence and Community-based QA. The scale of our dataset is similar to SO-DS (Heyman and Van Cutsem, 2020), CQA-Quora (Lyu et al., 2019) and CQA-SO (Zhang et al., 2021b). Our QA pairs are more complex than short-form query/code pairs, which resulting in less available resources on the web. Moreover, our dataset is the only cross-platform one.

B Baseline Implementation Details

We select GloVe average embedding, BERT $[CLS]$ token output and CodeBERT $[CLS]$ token output as baselines without supervision. For the supervised learning method, Siamese-formed CodeBERT and Triplet-formed CodeBERT are compared.

- BM25(okapi) (Robertson et al., 1995) BM25 is a well-known lexical retrieval model. We employ the implementation from the rank_bm25 library.5

- GloVe (Pennington et al., 2014) The mean embedding of the whole sentence is regarded as the representation of the sentence. Sentence representation similarity is computed for ranking.

- Universal Sentence Encoder (Cer et al., 2018) It is a transformer-based network which augments unsupervised learning with training on SNLI dataset. 6

- BERT (Kenton and Toutanova, 2019) We use the $[CLS]$ token output for sentence embedding.

- CodeBERT Similar to the strategy applied on BERT, the $[CLS]$ token output is adopted. Besides, we train a Siamese-formed CodeBERT and a Triplet-formed CodeBERT for comparison in supervised learning. The implementation of CosQA (Huang et al., 2021) is based on the Siamese-formed CodeBERT.

- S-BERT (Reimers and Gurevych, 2019) is a Siamese BERT-Networks. We employ the all-roberta-large-v1 model hosted on huggingface, which is pretrained over 1B+ QA pairs for better sentence embedding. 7

C Hyper-parameter Configuration

To figure out the different role that our proposed loss functions played in the optimization phrase, we conduct a discussion on weight hyper-parameters, $\alpha$, $\beta$, and $\gamma$. $\alpha$ and $\beta$ controls the importance of supervised contrastive loss over discrimination loss. Since $Cq$, $Cr$ and $D$ occurs together at the normalization part of equation 4.3, the larger their weight are, the more importance the have. We set $\alpha = \beta$ during our experiment, and report result as Table 7. We can find that the model performs better while $\alpha > 1$, which means the model focus more on separating elements from the same semantic category apart. The output demonstrates the importance and effectiveness of supervised contrastive learning. Besides, the weight $\gamma$ and the hard negative sampling rate $p$ are set to 0.2 to achieve best performance.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>MRR</th>
<th>P@1</th>
<th>P@5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>86.01</td>
<td>80.50</td>
<td>93.50</td>
</tr>
<tr>
<td>1.5</td>
<td>86.26</td>
<td>81.00</td>
<td>93.00</td>
</tr>
<tr>
<td>1</td>
<td>85.30</td>
<td>79.00</td>
<td>92.75</td>
</tr>
<tr>
<td>0.5</td>
<td>85.12</td>
<td>78.00</td>
<td>92.75</td>
</tr>
<tr>
<td>0</td>
<td>84.48</td>
<td>77.75</td>
<td>92.50</td>
</tr>
</tbody>
</table>

Table 7: Results of different hyperparameter $\alpha = \beta$ settings.

5https://GitHub.com/dorianbrown/rank_bm25
6https://tfhub.dev/google/universal-sentence-encoder/4
7https://huggingface.co/sentence-transformers/all-roberta-large-v1
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Domain</th>
<th>Size</th>
<th>Query Type</th>
<th>Answer Type</th>
<th>Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSN (Husain et al., 2019)</td>
<td>Coding</td>
<td>2.3M</td>
<td>Short description</td>
<td>Function code</td>
<td>No</td>
</tr>
<tr>
<td>Deep Code Search (Gu et al., 2018)</td>
<td>Coding</td>
<td>18.2M</td>
<td>Short description</td>
<td>Function code</td>
<td>No</td>
</tr>
<tr>
<td>CoSQA (Huang et al., 2021)</td>
<td>Coding</td>
<td>20.6K</td>
<td>Web Query</td>
<td>Function code</td>
<td>Yes</td>
</tr>
<tr>
<td>QECK (Nie et al., 2016)</td>
<td>Coding</td>
<td>312.9K</td>
<td>SO question title</td>
<td>Code block</td>
<td>No</td>
</tr>
<tr>
<td>StaQC (Yao et al., 2018)</td>
<td>Coding</td>
<td>268K</td>
<td>SO question title</td>
<td>Code block</td>
<td>Partly</td>
</tr>
<tr>
<td>CoNaLa (Yin et al., 2018)</td>
<td>Coding</td>
<td>598.2K</td>
<td>SO question title</td>
<td>Code block</td>
<td>Partly</td>
</tr>
<tr>
<td>SO-DS (Heyman and Van Cutsem, 2020)</td>
<td>Coding</td>
<td>12.1k</td>
<td>SO question title</td>
<td>Code block</td>
<td>No</td>
</tr>
<tr>
<td>CQA-Quora (Lyu et al., 2019)</td>
<td>Open</td>
<td>76.2k</td>
<td>Quora question</td>
<td>Quora Answer</td>
<td>No</td>
</tr>
<tr>
<td>CQA-SO (Zhang et al., 2021b)</td>
<td>Coding</td>
<td>13.9k</td>
<td>SO question</td>
<td>SO Answer</td>
<td>No</td>
</tr>
<tr>
<td>Repo4QA (ours)</td>
<td>Coding</td>
<td>13.0k</td>
<td>SO question</td>
<td>GitHub repository</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 6: Overview of existing datasets on Code Intelligence and Community-based QA