Adversarial Training for Code Retrieval with Question-Description Relevance Regularization

Jie Zhao
The Ohio State University
zhao.1359@osu.edu

Huan Sun
The Ohio State University
sun.397@osu.edu

Abstract

Code retrieval is a key task aiming to match natural and programming languages. In this work, we propose adversarial learning for code retrieval, that is regularized by question-description relevance. First, we adapt a simple adversarial learning technique to generate difficult code snippets given the input question, which can help the learning of code retrieval that faces bi-modal and data-scarce challenges. Second, we propose to leverage question-description relevance to regularize adversarial learning, such that a generated code snippet should contribute more to the code retrieval training loss, only if its paired natural language description is predicted to be less relevant to the user given question. Experiments on large-scale code retrieval datasets of two programming languages show that our adversarial learning method is able to improve the performance of state-of-the-art models. Moreover, using an additional duplicate question prediction model to regularize adversarial learning further improves the performance, and this is more effective than using the duplicated questions in strong multi-task learning baselines.[1]

1 Introduction

Recently there has been a growing research interest in the intersection of natural language (NL) and programming language (PL), with exemplar tasks including code generation (Agashe et al., 2019; Bi et al., 2019), code summarizing (LeClair and McMillan, 2019; Panthaplackel et al., 2020), and code retrieval (Gu et al., 2018). In this paper, we study code retrieval, which aims to retrieve code snippets for a given NL question such as *"Flatten a shallow list in Python."* Advanced code retrieval tools can save programmers tremendous time in various scenarios, such as how to fix a bug, how to implement a function, which API to use, etc. Moreover, even if the retrieved code snippets do not perfectly match the NL question, editing them is often much easier than generating a code snippet from scratch. For example, the retrieve-and-edit paradigm (Hayati et al., 2018; Hashimoto et al., 2018; Guo et al., 2019) for code generation has attracted growing attention recently, which first employs a code retriever to find the most relevant code snippets for a given question, and then edit them via a code generation model. Previous work has shown that code retrieval performance can significantly affect the final generated results (Huang et al., 2018) in such scenarios.

There have been two groups of work on code retrieval: (1) One group of work (e.g., the recent retrieve-and-edit work (Hashimoto et al., 2018; Guo et al., 2019)) assumes each code snippet is associated with NL descriptions and retrieves code snippets by measuring the relevance between such descriptions and a given question. (2) The other group of work (e.g., CODENN (Iyer et al., 2016) and Deep Code Search (Gu et al., 2018)) directly measures the relevance between a question and a code snippet. Comparing with the former group, this group of work has the advantage that they can still apply when NL descriptions are not available for candidate code snippets, as is often the case for many large-scale code repositories (Dinella et al., 2020; Chen and Monperrus, 2019). Our work connects with both groups: We aim to directly match a code snippet with a given question, but during training, we will utilize question-description relevance to improve the learning process.

Despite the existing efforts, we observe two challenges for directly matching code snippets with NL questions, which motivate this work. First, code retrieval as a bi-modal task requires representation learning of two heterogeneous but complementary

[1]Source code and dataset are available at https://github.com/jiez-osu/QQC.

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 4049–4059

modalities, which has been known to be difficult (Cvitkovic et al., 2019; LeC; Akbar and Kak, 2019) and may require more training data. This makes code retrieval more challenging compared to document retrieval where the target documents often contain useful shallow NL features like keywords or key phrases. Second, code retrieval often encounters special one-to-many mapping scenarios, where one NL question can be solved by multiple code solutions that take very different approaches. Table 1 illustrates the challenges. For i=1,2 or 3, $q^{(i)}$ is an NL question/description that is associated with a Python answer $c^{(i)}$. Here, question $q^{(1)}$ should be matched with multiple code snippets: $c^{(1)}$ and $c^{(2)}$, because they both flatten a 2D list despite with different programming approaches. In contrast, $c^{(3)}$ is performing a totally different task, but uses many overlapped tokens with $c^{(1)}$. Hence, it can be difficult to train a code retrieval model that generalizes well to match $q^{(1)}$ with both $c^{(1)}$ and $c^{(2)}$, and is simultaneously able to distinguish $c^{(1)}$ from $c^{(3)}$.

To address the first challenge, we propose to introduce adversarial training to code retrieval, which has been successfully applied to transfer learning from one domain to another (Tzeng et al., 2017) or learning with scarce supervised data (Kim et al., 2019). Our intuition is that by employing a generative adversarial model to produce *challenging negative code snippets* during training, the code retrieval model will be strengthened to distinguish between positive and negative $\langle q, c\rangle$ pairs. In particular, we adapt a generative adversarial sampling technique (Wang et al., 2017), whose effectiveness has been shown in a wide range of uni-modal text retrieval tasks.

For the second challenge, we propose to further employ *question-description (QD) relevance* as a complementary uni-modal view to reweight the adversarial training samples. In general, our intuition is that the code retrieval model should put more weights on the adversarial examples that are hard to distinguish by itself, but easy from the view of a QD relevance model. This design will help solve the one-to-many issue in the second challenge, by differentiating true negative and false negative adversarial examples: If a QD relevance model also suggests that a code snippet is not relevant to the original question, it is more likely to be a true negative, and hence the code retrieval model should put more weights on it. Note that this QD relevance design aims to help train the code retrieval model better and we do not need NL descriptions to be associated with code snippets at testing phase.

$q^{(1)}$	*Flatten a shallow list in Python*
$c^{(1)}$	`from itertools import chain` `rslt = chain(*list_2d)`
$q^{(2)}$	*How to flatten a 2D list to 1D without using numpy?*
$c^{(2)}$	`list_of_lists = [[1,2,3],[1,2],[1,4,5,6,7]]` `[j for sub in list_of_lists for j in sub]`
$q^{(3)}$	*How to get all possible combinations of a list's elements?*

$c^{(3)}$	`from itertools import chain, combinations`
	`subsets = chain(*map(lambda x: combinations(`
	`mylist, x), range(0, len(mylist)+1)))`

Table 1: Motivating Example. $\langle q^{(i)}, c^{(i)}\rangle$ denotes an associated ⟨natural language question, code snippet⟩ pair. $q^{(i)}$ can also be viewed as a description of $c^{(i)}$. Given $q^{(1)}$, the ideal code retrieval result is to return both $c^{(1)}$ and $c^{(2)}$ as their programming semantics are equivalent. Contrarily, $c^{(3)}$ is semantically irrelevant to $q^{(1)}$ and should not be returned, although its surface form is similar to $c^{(1)}$. In such cases, it can be easier to decide their relationships from the question perspective, because $\langle q^{(1)}, q^{(2)}\rangle$ are more alike than $\langle q^{(1)}, q^{(3)}\rangle$.

We conduct extensive experiments using a large-scale ⟨question, code snippet⟩ dataset StaQC (Yao et al., 2018) and our collected duplicated question dataset from Stack Overflow[2]. The results show that our proposed learning framework is able to improve the state-of-the-art code retrieval models and outperforms using adversarial learning without QD relevance regularization, as well as strong multi-task learning baselines that also utilize question duplication data.

2 Overview

The work studies ***code retrieval***, a task of matching questions with code, which we will use **QC** to stand for. The training set $\mathcal{D}^{\text{QC}}$ consists of NL question and code snippet pairs $\mathcal{D}^{\text{QC}}=\{q^{(i)}, c^{(i)}\}$. Given NL question $q^{(i)}$, the QC task is to find $c^{(i)}$ from $\mathcal{D}^{\text{QC}}$ among all the code snippets. For simplicity, we omit the data sample index and use q and c to denote a QC pair, and c^{-} to represent any other code snippets in the dataset except for c.

Our goal is to learn a QC model, denoted as f_θ^{QC}, that retrieves the highest score code snippets for an input question: $\arg\max_{c'\in\{c\}\cup\{c^-\}} f_\theta^{\text{QC}}(q, c')$. Note that at testing time, the trained QC model f^{QC} can be used to retrieve code snippets from any code bases, unlike the group of QC methods (Hayati et al., 2018; Hashimoto et al., 2018; Guo et al.,

[2] https://stackoverflow.com/

2019) relying on the availability of NL descriptions of code.

We aim to address the aforementioned challenges in code retrieval through two strategies: (1) We introduce adversarial learning (Goodfellow et al., 2014a) to alleviate the bi-modal learning challenges. Specifically an adversarial QC generator selects unpaired code snippets that are difficult for the QC model to discriminate, to strengthen its ability to distinguish top-ranked positive and negative samples (Wang et al., 2017). (2) We also propose to employ a question-description (**QD**) relevance model to provide a secondary view on the generated adversarial samples, inspired by the group of QC work that measures the relevance of code snippets through their associated NL descriptions.

Figure 1 gives an overview of our proposed learning framework, which does not assume specific model architectures and can be generalized to different base QC models or use different QD relevance models. A general description is given in the caption. In summary, the adversarial QC generator selects $\hat{c}$ that is unpaired with a given q. $\hat{q}$ is an NL description of $\hat{c}$. Details on how to acquire $\hat{q}$ will be introduced in Section 3.2. Next, a QD model predicts a relevance score for $\langle q, \hat{q} \rangle$. A pairwise ranking loss is calculated based on whether the QC model discriminates ground-truth QC pair $\langle q, c \rangle$ from unpaired $\langle q, \hat{c} \rangle$. Learning through this loss is reweighted by a down-scale factor, which is dynamically determined by the QD relevance prediction score. This works as a regularization term over potential false negative adversarial samples.

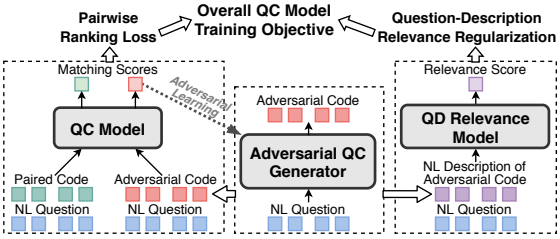

Figure 1: Regularized adversarial learning framework. Best viewed in color. The adversarial QC generator (middle) produces an adversarial code given an NL question. The QD relevance model (right) then predicts a relevance score between the given question and the NL description or the generated adversarial code. A pairwise ranking loss is computed between the ground-truth code and the adversarial code. The QC model (left) is trained with the ranking loss, after it is scaled by a QD relevance regularization weight that depends on the QD relevance score. The parameter update is larger when the relevance score is smaller and vice versa.

3 Proposed Methodology

We now introduce in detail our proposed learning framework. We start with the adversarial learning method in Section 3.1 and then discuss the rationale to incorporate question-description or QD relevance feedback in Section 3.2, before putting them together in Section 3.3 and Section 3.4.

3.1 Adversarial Learning via Sampling

We propose to apply adversarial learning (Goodfellow et al., 2014a) to code retrieval. Our goal is to train a better QC model f_θ^{QC} by letting it play the adversarial game with a QC generator model g_ϕ^{QC}. θ represents the parameters of the QC model and ϕ represents the parameters of the adversarial QC generator. As in standard adversarial learning, f_θ^{QC} plays the discriminator role to distinguish ground-truth code snippet c from generated pairs $\hat{c}$. The training objective of the QC model is to minimize $\mathcal{L}_\theta$ below:

$$\mathcal{L}_\theta = \sum_i \mathbb{E}_{\hat{c} \sim P_\phi(c|q^{(i)})} l_\theta(q^{(i)}, c^{(i)}, \hat{c}),$$

$$l_\theta = \max(0, d + f_\theta^{\text{QC}}(q^{(i)}, \hat{c}) - f_\theta^{\text{QC}}(q^{(i)}, c^{(i)})),$$

where l_θ is a pairwise ranking loss, and specifically we use a hinge loss with margin d. $\hat{c}$ is generated by g_ϕ^{QC} and follows a probability distribution $P_\phi(c|q^{(i)})$. g_ϕ^{QC} aims to assign higher probabilities to code snippets that would mislead f_θ^{QC}.

There are many ways to realize the QC generator. For example, one may employ a sequence model to generate the adversarial code snippet $\hat{c}$ token by token (Bi et al., 2019; Agashe et al., 2019). However, training a sequence generation model is difficult, because the search space of all code token combinations is huge. Henceforce, we turn to a simpler idea inspired by Wang et al. (2017), and restrict the generation of $\hat{c}$ to the space of all the existing code snippets in the training dataset $\mathcal{D}^{\text{QC}}$. The QC generator then only needs to sample an existing code snippet $c^{(j)}$ from an adversarial probability distribution conditioned on a given query and let it be $\hat{c}$, i.e., $\hat{c}{=}c^{(j)}{\sim}P_\phi(c|q^{(i)})$. Adopting this method will make training the QC generator easier, and ensures that the generated code snippets are legitimate as they directly come from the training dataset. We

define the adversarial code distribution as:

$$P_\phi(c|q^{(i)}) = \frac{\exp(g_\phi^{\text{QC}}(q^{(i)}, c)/\tau)}{\sum_{c'} \exp(g_\phi^{\text{QC}}(q^{(i)}, c')/\tau)},$$

where g_ϕ^{QC} represents an adversarial QC matching function. τ is a temperature hyper-parameter used to tune the distribution to concentrate more of less on top-scored code snippets. Moreover, scoring all code snippets can be computationally inefficient in practice. Therefore, we use the method of Yang et al. (2019) to first uniformly sample a subset of data, whose size is much smaller than the entire training set size, and then perform adversarial sampling on this subset.

The generator function g_ϕ^{QC} can be pre-trained in the same way as the discriminator (i.e., f_θ^{QC}) and then get updated using standard policy gradient reinforcement learning algorithms, such as REINFORCE (Williams, 1992), to maximize the ranking losses of the QC model. Formally, the QC generator aims to maximize the following expected reward: $J(\phi)=\sum_i \mathbb{E}_{c^{(j)}\sim P_\phi(c|q^{(i)})}[l_\theta(q^{(i)}, c^{(i)}, c^{(j)})]$, where $l_\theta(q^{(i)}, c^{(i)}, c^{(j)})$ is the pairwise ranking loss of the discriminator model defined earlier. The gradient of J can be derived as $\nabla_\phi J=\sum_i \mathbb{E}_{c^{(j)}\sim P_\phi(c|q^{(i)})}[l_\theta \cdot \nabla_\phi \, log P_\phi(c^{(j)}|q^{(i)})]$. Another option is to let g_ϕ^{QC} use the same architecture as f_θ^{QC} and use tied parameters (i.e., $\phi=\theta$), as adopted in previous work (Deshpande and M.Khapra, 2019; Park and Chang, 2019).

The focus of this work is to show the effectiveness of applying adversarial learning to code retrieval, and how to regularize it with QD relevance. We leave more complex adversarial techniques (e.g. adversarial perturbation (Goodfellow et al., 2014b; Miyato et al., 2015) or adversarial sequence generation (Li et al., 2018)) for future studies.

3.2 Question-Description Relevance Regularization

Intuitively, we can train a better code retrieval model, if the negative code snippets are all true-negative ones, i.e., if they are confusingly similar to correct code answers, but perform different functionalities. However, because of the one-to-many mapping issue, some negative code snippets sampled by the adversarial QC generator can be false-negative, i.e. they are equally good answers for a given question despite that they are not paired with the question in the training set. Unfortunately during training, this problem could become increasingly obvious as the adversarial will be improved along with the code retrieval model, and eventually makes learning less and less effective. Since both the QC model and the adversarial QC generator operates from the QC perspective, it is difficult to further discriminate true-negative and false-negative code snippets.

Therefore, we propose to alleviate this problem with QD relevance regularization. This idea is inspired by the group of QC work mentioned in Section 1 that retrieves code snippets by matching their NL descriptions with a given question. But different from them, we only leverage QD relevance during training to provide a secondary view and to reweight the adversarial samples. Fortunately, an adversarial code snippet $\hat{c}$ sampled from the original training dataset $\mathcal{D}^{\text{QC}}$ is paired with an NL question $\hat{q}$, which can be regarded as its NL description and used to calculate the relevance to the given question q.

Let us refer to the example in Table 1 again. At a certain point of training, with $q^{(1)}$ "*Flatten a shallow list in Python*" being the given question, the adversarial QC generator may choose $c^{(2)}$ and $c^{(3)}$ as the negative samples, but instead of treating them equivalently, we can infer from the QD matching perspective that $c^{(3)}$ is likely to be true negative, because $q^{(3)}$ "*How to get all possible combinations of a list's elements*" clearly has different meanings from $q^{(1)}$, while $c^{(2)}$ is likely to be a false negative example since $q^{(2)}$ "*How to flatten a 2D list to 1D without using numpy?*" is similar to $q^{(1)}$. Hence, during training, the discriminative QC model should put more weights on negative samples like $c^{(3)}$ rather than $c^{(2)}$.

We now explain how to map QD relevance scores to regularization weights. Let $f^{\text{QD}}(q, \hat{q})$ denote the predicted relevance score between the given question q and the question paired with an adversarial code snippet $\hat{q}$, and let $f^{\text{QD}}(q, \hat{q})$ be normalized to the range from 0 to 1. We can see from the above example that QD relevance and adjusted learning weight should be reversely associated, so we map the normalized relevance score to a weight using a monotonously decreasing polynomial function: $w^{\text{QD}}(x)=(1-x^a)^b$, $0\leq x\leq 1$. Both a and b are positive integer hyper-parameters that control the shape of the curve and can be tuned on the dev sets. In this work, they are both set to one by default for simplicity. $w^{\text{QD}} \in [0, 1]$ allows the optimization objective to weigh less on adversarial samples that

Algorithm 1: Question-Description Relevance Regularized Adversarial Learning.

QC training data : $\mathcal{D}^{\text{QC}} = \{q^{(i)}, c^{(i)}\}$
QD model : f^{QD}
Constants : positive intergers N, τ, a, b
Result: QC model f_θ^{QC}

1 ▷ Pretrain f_θ^{QC} on $\mathcal{D}^{\text{QC}}$ using pairwise ranking loss l_θ^{QC} with randomly sampled negative code snippets ;
2 ▷ Initialize QC generator g_ϕ^{QC} with f_θ^{QC}: $\phi \leftarrow \theta$;
3 **while** *not converge or not reach max iter number* **do**
4 **for** *random sampled* $\langle q^{(i)}, c^{(i)} \rangle \in \mathcal{D}^{QC}$ **do**
5 Randomly choose D=$\{q$,$c\} \subset \mathcal{D}^{\text{QC}}$, where $|D|$=N;
6 Sample $c^{(j)} \in D$, that $c^{(j)} \sim P_\phi(c^{(j)}|q^{(i)}) = \text{softmax}_\tau(g_\phi^{\text{QC}}(q^{(i)}, c^{(j)}))$;
7 $l_\theta^{\text{QC}} \leftarrow l_\theta(q^{(i)}, c^{(i)}, c^{(j)})$;
8 Find $q^{(j)}$ associated with $q^{(j)}$, $w^{\text{QD}} \leftarrow (1 - f^{\text{QD}}(q^{(i)}, q^{(j)})^a)^b$;
9 Update QC model with gradient descent to reduce loss: $w^{\text{QD}} \cdot l_\theta^{\text{QC}}$;
10 Update adversarial QC generator with gradient ascent: $l_\theta^{\text{QC}} \cdot \nabla_\phi \, logP_\phi(c^{(j)}|q^{(i)})$
11 **end**
12 ▷ Optional QD model update. (See Section 3.4.)
13 **end**

are more likely to be false negative.

3.3 Question-Description Relevance Regularized Adversarial Learning

Now we describe the proposed learning framework in Algorithm 1 that combines adversarial learning and QD relevance regularization. Let us first assume the QD model is given and we will explain how to pre-train, and optionally update it shortly.

The QC model can be first pre-trained on $\mathcal{D}^{\text{QC}}$ using standard pairwise ranking loss $l_\theta(q^{(i)}, c^{(i)}, c^{(j)})$ with randomly sampled $c^{(j)}$. Line 3-11 show the QC model training steps. For each QC pair $\langle q^{(i)}, c^{(i)} \rangle$, a batch of negative QC pairs are sampled randomly from the training set $\mathcal{D}^{\text{QC}}$. The QC generator then choose an adversarial $c^{(j)}$ from distribution $P_\phi(c|q^{(i)})$ defined in Section 3.1, and its paired question is $q^{(j)}$. Two questions $q^{(i)}$ and $q^{(j)}$ are then passed to the QD model, and the QD relevance prediction is mapped to a regularization weight w^{QD}. Finally, the regularization weight is used to control the update of the QC model on the ranking loss with the adversarial $\hat{c}$.

3.4 Base Model Architecture

Our framework can be instantiated with various model architectures for QC or QD. Here we choose the same neural network architecture as (Gu et al., 2018; Yao et al., 2019) as our base QC model, that achieves competitive or state-of-the-art code retrieval performances. Concretely, both a natural language question q and a code snippet c are sequences of tokens. They are encoded respectively by separate bi-LSTM networks (Schuster and Paliwal, 1997), passed through a max pooling layer to extract the most salient features of the entire sequence, and then through a hyperbolic tangent activate function. The encoded question and code representations are denoted as h^q and h^c. Finally, a matching component scores the vector representation between q and c and outputs their matching score for ranking. We follow previous work to use cosine similarity: $f^{\text{QC}}(q, c) = \text{cosine}(h^q, h^c)$.

QD Model. There are various model architecture choices, but here for simplicity, we adapt the QC model for QD relevance prediction. We let the QD model use the same neural architecture as the QC model, but with Siamese question encoders. The QD relevance score is the cosine similarity between $h^{q^{(i)}}$ and $h^{q^{(j)}}$, the bi-LSTM encoding outputs for question $q^{(i)}$ and $q^{(j)}$ respectively: $f^{\text{QD}}(q^{(i)},q^{(j)})$=cosine$(h^{q^{(i)}},h^{q^{(j)}})$. This method allows using a pre-trained QC model to initialize the QD model parameters, which is easy to implement and the pre-trained question encoder in the QC model can help the QD performance. Since programming-domain question paraphrases are rare, we collect a small QD training set consisting of programming related natural language question pairs $\mathcal{D}^{\text{QD}}$=$\{q^{(j)}, p^{(j)}\}$ based on duplicated questions in Stack Overflow.

The learning framework can be symmetrically applied, as indicated by Line 12 in Algorithm 1, so that the QD model can also be improved. This may provide better QD relevance feedback to help train a better QC model. In short, we can use a discriminative and a generative QD model. The generative QD model selects adversarial questions to help train the discriminative QD model, and this training can be regularized by the relevance predictions from a QC model. More details will be introduced in the experiments.

4 Experiments

In this section, we first introduce our experimental setup, and then will show that our method not only outperforms the baseline methods, but also multi-task learning approaches, where question-description relevance prediction is the other task. In

	Python			SQL		
	Train	**Dev**	**Test**	**Train**	**Dev**	**Test**
QC	68,235	8,529	8,530	60,509	7,564	7,564
QD	1,085	1,085	1,447	18,020	2,252	2,253

Table 2: Dataset statistics. QD is used to represent the duplicate question dataset.

particular, the QD relevance regularization consistently improves QC performance upon adversarial learning, and the effectiveness of relevance regularization can also be verified as it is symmetrically applied to improve the QD task.

4.1 Datasets

We use StaQC (Yao et al., 2018) to train and evaluate our code retrieval model, which contains automatically extracted questions on Python and SQL and their associated code answers from Stack Overflow. We use the version of StaQC that each question is associated with a single answer, as those associated with multiple answers are predicted by an automatic answer detection model and therefore noisier. We randomly split this QC datasets by a 70/15/15 ratio into training, dev and testing sets. The dataset statistics are summarized in Table 2.

We use Stack Exchange Data Explorer[3] to collect data for training and evaluating QD relevance prediction. Specifically, we collect the question pairs from posts that are manually labeled as duplicate by users, which are related by `LinkTypeId=3`. It turns out that the QD datasets are substantially smaller than the QC datasets, especially for Python, as shown in Table 2. This makes it more interesting to check whether a small amount of QD relevance guidance can help improve code retrieval performances.

4.2 Baselines and Evaluation Metrics

We select state-of-the-art methods from both groups of work for QC (mentioned in Introduction). DecAtt and DCS below are methods that directly match questions with code. EditDist and vMF-VAE transfer code retrieval into a question matching problem.

- DecAtt (Parikh et al., 2016). This is a widely used neural network model with attention mechanism for sentence pairwise modeling.
- DCS (Gu et al., 2018). We use this as our base model, because it is a simple yet effective code retrieval model that achieves competitive performance without introducing additional training overheads (Yao et al., 2019). Its architecture has been described in Section 3.4.
- EditDist (Hayati et al., 2018). Code snippets are retrieved by measuring an edit distance based similarity function between their associated NL descriptions and the input questions. Since there is only one question for each sample in the QC datasets, we apply a standard code summarization tool (Iyer et al., 2016) to generate code descriptions to match with input questions.
- vMF-VAE (Guo et al., 2019). This is similar to EditDist, but a vMF Variational Autoencoder (Xu and Durrett, 2018) is separately trained to embed questions and code descriptions into latent vector distributions, whose distance is then measured by KL-divergence. This method is also used by Hashimoto et al. (2018).

We further consider multi-task learning (MTL) as an alternative way how QD can help QC. It is worth mentioning that our method does not require *associated* training data or the sharing of trained parameters between the QD and QC tasks, whereas MTL typically does. For fair comparison, we adapt two MTL methods to our scenario that use the same base model, or its question and code encoders:

- MTL-DCS. This is a straightfoward MTL adaptation of DCS, where the code encoder is updated on the QC dataset and the question encoder is updated on both QC and QD datasets. The model is alternatively trained on both datasets.
- MTL-MLP (Gonzalez et al., 2018). This recent MTL method is originally designed to rank relevant questions and question-related comments. It uses a multi-layer perceptron (MLP) network with one shared hidden layer, a task-specific hidden layer and a task-specific classification layer for each output. We adapt it for our task. The input to the MLP is the concatenation of similarity features $[max(h^q, h^c), h^q - h^c, h^q \odot h^c]$, where $\odot$ is element-wise product. h^q and h^c are learned using the same encoders as our base model.

The ranking metrics used for evaluation are Mean Average Precision (MAP) and Normalize Discounted Cumulative Gain (nDCG) (Järvelin and Kekäläinen, 2002). The same evaluation method as previous work is adopted (Iyer et al., 2016; Yao et al., 2019) for both QC and QD, where we randomly choose from the testing set a fixed-size (49) pool of negative candidates for each question, and

[3] SEDE and SEDE schema documentation.

	Python		SQL	
	MAP	**nDCG**	**MAP**	**nDCG**
EditDist (Hayati et al., 2018)	0.2348	0.3844	0.2096	0.3641
vMF-VAE (Guo et al., 2019)	0.2886	0.4511	0.2921	0.4537
DecAtt (Parikh et al., 2016)	0.5744	0.6716	0.5142	0.6231
DCS (Gu et al., 2018)	0.6015	0.6929	0.5155	0.6237
MTL-MLP (Gonzalez et al., 2018)	0.5737	0.6712	0.5079	0.6179
MTL-DCS	0.6024	0.6935	0.5160	0.6237
Our	**0.6372***	**0.7206***	**0.5404***	**0.6429***
Our - RR	0.6249*	0.7111*	0.5274*	0.6327*

Table 3: Code retrieval (QC) performance on test sets. * denotes significantly different from DCS (Gu et al., 2018) in one-tailed t-test ($p < 0.01$).

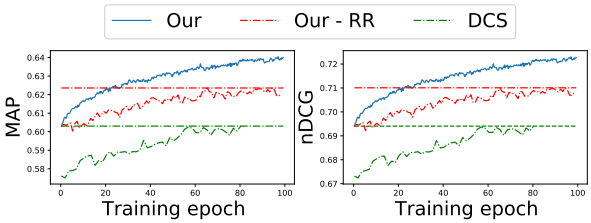

Figure 2: QC learning curves on the Python dev set.

evaluate the ranking of its paired code snippet or questions among these negative candidates.

4.3 Implementation Details

Our implementation is based on Yao et al. (2019). We follow this work to set the base model hyper-parameters. The vocabulary embedding size for both natural language and programming language is set at 200. The LSTM hidden size is 400. Margin in the hinge loss is 0.05. The trained DCS model is used as pre-training for our models. The learning rate is set at 1e-4 and the dropout rate set at 0.25. For adversarial training, we set τ to 0.2 following (Wang et al., 2017) and limit the maximum number of epochs to 300. Standard L2-regularization is used on all the models. We empirically tried to tie the parameters of the discriminator and the generator following previous work (Deshpande and M.Khapra, 2019; Park and Chang, 2019), which shows similar improvements over the baselines. Implementation from Xu and Durrett (2018) is used for the vMF-VAE baseline.

We follow the code preprocessing steps in Yao et al. (2018) for Python and Iyer et al. (2016) for SQL. We use the NLTK toolkit (Bird and Loper, 2004) to tokenize the collected duplicate questions, and let it share the same NL vocabulary as the QC dataset $\mathcal{D}^{\text{QC}}$.

4.4 Results and Analyses

Our experiments aim to answer the following research questions:

(1) *Can the question regularized adversarial learning framework improve code retrieval (QC) performance?* We will first compare the code retrieval performance of different methods. Table 3 summarizes the *test results*, which are consistent on both Python and SQL datasets. Code retrieval baselines by measuring QD relevance, e.g., EditDist and vMF-VAE, are popularly used in code generation related work, but do not perform well compared to other code retrieval baselines in our experiments, partly because they are not optimized toward the QC task. This suggests that applying more advanced code retrieval methods for retrieve-and-edit code generation can be an interesting future research topic. DCS is a strong baseline, as it outperforms DecAtt that uses a more complex attention mechanism. This indicates that it is not easy to automatically learn pairwise token associations between natural language and programming languages from software community data, which is also suggested by previous work (Panthaplackel et al., 2019; Vinayakarao et al., 2017).

Our proposed learning algorithm can improve the QC performance compared to all the baselines. The "- RR" variant is to only apply adversarial sampling without QD relevance regularization. It already leads to improvements compared to the base model (i.e. DCS), but does not perform as well as our full model. This proves the usefulness of the QD relevance regularization and indicates that selectively weighting the contribution of adversarial samples to the training loss can help the model generalize better to test data. Figure 2 compares QC learning curves on the *dev set*. The full model curve being the smoothest qualitatively suggests that the adversarial learning has been well regularized.

(2) *How does the proposed algorithm compare with multi-task learning methods?* The results are reported in Table 4. The MTL-MLP model is originally proposed to improve question-question relevance prediction by using question-comment relevance prediction as a secondary task (Gonzalez et al., 2018). It does not perform as well as MTL-DCS, which basically uses hard parameter sharing between the two tasks and does not require additional similarity feature definitions. In general, the effectiveness of these MTL baselines on the QC task is limited because there are only a small amount of QD pairs available for training. Both our method and its ablated variant outperform the

	Python		SQL	
	MAP	**nDCG**	**MAP**	**nDCG**
MTL-MLP (Gonzalez et al., 2018)	0.5737	0.6712	0.5079	0.6179
MTL-DCS	0.6024	0.6935	0.5160	0.6237
Our	**0.6372**	**0.7206**	**0.5404**	**0.6429**

Table 4: Compare QC performance with MTL.

MTL baselines. This shows that it may be more effective to use a data scarce task to regularize the adversarial learning of a relatively data rich task, than using those scarce data in MTL.

(3) *Can the QD performance be improved by the proposed method?* Although QD is not the focus of this work, we can use it to verify that generalizability of our method by symmetrically applying it to update the QD model as mentioned in Section 3.2. To be concrete, a generative adversarial QD model selects difficult questions from the a distribution of question pair scores: $\hat{q} \sim \text{softmax}_\tau(f^{\text{QD}}(\hat{q}, q^{(i)}))$. Then a QC model is used to calculate a relevance score for a question-code pair, and this can regularize the adversarial learning of the QD model.

Table 5 shows the results. Our method and its ablated variants outperform the QD baselines EditDist and vMF-VAE, again suggesting that supervised learning is more effective. The full model achieves the best overall performance and removing relevance regularization (- RR) from the QC model consistently leads to performance drop. In contrast, further removing adversarial sampling (- AS) hurts the performance on SQL dataset slightly, but not on Python. This is probably because the Python QD dataset is very small and using adversarial learning can easily overfit, which again suggests the importance of our proposed relevance regularization. Finally, removing QC as pretraining (- Pretrain) greatly hurts the performance, which is understandable since QC datasets are much larger.

Because the QD model performance can be improved in such a way, we allow it to get updated in our QC experiments (corresponding to line 12 in Algorithm 1) and the results have been discussed in Table 3. We report here the QC performance using a fixed QD model (i.e. Our - RR - AS) for relevance regularization: MAP=0.6371, nDCG=0.7205 for Python and MAP=0.5366, nDCG=0.6398 for SQL. Comparing these results with those in Table3 (Our), one can see that allowing the QD model to update consistently improves QC performance, which suggests that a better QD model can provide more accurate relevance regularization to the QC model and leads to better results.

	Python		SQL	
	MAP	**nDCG**	**MAP**	**nDCG**
EditDist (Hayati et al., 2018)	0.3617	0.4883	0.3246	0.4580
vMF-VAE (Guo et al., 2019)	0.3009	0.4616	0.3029	0.4641
Our	**0.7162**	**0.7821**	**0.6947**	**0.7651**
Our - RR	0.7046	0.7734	0.6846	0.7575
Our - RR - AS	0.7116	0.7787	0.6764	0.7512
Our - RR - AS - Pretrain	0.3882	0.5170	0.6284	0.7129

Table 5: Question relevance prediction results, evaluated on the question duplication dataset we collected.

5 Related Work

Code Retrieval. Code retrieval has developed from using classic information retrieval techniques (Hill et al., 2014; Haiduc et al., 2013; Lu et al., 2015) to recently deep neural methods that can be categorized into two groups. The first group directly model the similarity across the natural language and programming language modalities. Besides CODENN (Iyer et al., 2016) and DCS (Gu et al., 2018) discussed earlier, Yao et al. (2019) leverage an extra code summarization task and ensemble a separately trained code summary retrieval model with a QC model to achieve better overall code retrieval performances. Ye et al. (2020) further train a code generation model and a code summarization model through dual learning, which helped to learn better NL question and code representations. Both works employ additional sequence generation models that greatly increases the training complexity, and they both treat all unpaired code equally as negatives. Our work differs from them as we introduce adversarial learning for code retrieval, and the existing work do not leverage question relevance for code retrieval as we do.

The second group of works transfer code retrieve to a code description retrieval problem. This methodology has been widely adopted as a component in the retrieve-and-edit code generation literature. For example, heuristic methods such as measuring edit distance (Hayati et al., 2018) or comparing code type and length (Huang et al., 2018) are used, and separate question latent representations (Hayati et al., 2018; Guo et al., 2019) are learned. Our work shares with them the idea to exploit QD relevance, but we use QD relevance in a novel way to regularize the adversarial learning of QC models. It will be an interesting future work to leverage the proposed code retrieval method for retrieve-and-edit code generation.

Adversarial Learning. Adversarial learning has been widely used in areas such as computer vision

(Mirza and Osindero, 2014; Chen et al., 2016; Radford et al., 2015; Arjovsky et al., 2017), text generation (Yu et al., 2017; Chen et al., 2019; Liang, 2019; Gu et al., 2018; Liu et al., 2017; Ma et al., 2019), relation extraction (Wu et al., 2017; Qin et al., 2018), question answering (Oh et al., 2019; Yang et al., 2019), etc. We proposed to apply adversarial learning to code retrieval, because they have effectively improved cross-domain task performances and helped generate useful training data, We adapted the method from Wang et al. (2017) for the bi-modal QC scenario. As future work, adversarial learning for QC can be generalized to other settings with different base neural models (Yang et al., 2019) or with more complex adversarial learning methods, such as adding perturbed noises (Park and Chang, 2019) or generating adversarial sequences (Yu et al., 2017; Li et al., 2018). Our method differs from most adversarial learning work in that the discriminator (QC model) does not see all generated samples as equally negative.

6 Conclusion

This work studies the code retrieval problem, and tries to tackle the challenges of matching natural language questions with programming language (code) snippets. We propose a novel learning algorithm that introduces adversarial learning to code retrieval, and it is further regularized from the perspective of a question-description relevance prediction model. Empirical results show that the proposed method can significantly improve the code retrieval performances on large-scale datasets for both Python and SQL programming languages.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments. This research was sponsored in part by the Army Research Office under cooperative agreements W911NF-17-1-0412, NSF Grant IIS1815674, and NSF CAREER #1942980. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

References

Rajas Agashe, Srini Iyer, and Luke Zettlemoyer. 2019. Juice: A large scale distantly supervised dataset for open domain context-based code generation. *ArXiv*, abs/1910.02216.

Shayan A. Akbar and Avinash C. Kak. 2019. Scor: Source code retrieval with semantics and order. *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 1–12.

Martín Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein generative adversarial networks. In *ICML*.

Bin Bi, Chen Wu, Ming Yan, Wei Wang, Jiangnan Xia, and Chenliang Li. 2019. Incorporating external knowledge into machine reading for generative question answering. *ArXiv*, abs/1909.02745.

Steven Bird and Edward Loper. 2004. NLTK: The natural language toolkit. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 214–217, Barcelona, Spain. Association for Computational Linguistics.

MK Chen, Xinyi Lin, Chen Wei, and Rui Yan. 2019. Bofgan: Towards a new structure of backward-or-forward generative adversarial nets. In *The World Wide Web Conference*, pages 2652–2658. ACM.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. 2016. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NIPS*.

Zimin Chen and Martin Monperrus. 2019. A literature study of embeddings on source code. *ArXiv*, abs/1904.03061.

Milan Cvitkovic, Badal Singh, and Anima Anandkumar. 2019. Open vocabulary learning on source code with a graph-structured cache. In *ICML*.

Ameet Deshpande and Mitesh M.Khapra. 2019. Dissecting an adversarial framework for information retrieval.

Elizabeth Dinella, Hanjun Dai, Ziyang Li, Mayur Naik, Le Song, and Ke Wang. 2020. Hoppity: Learning graph transformations to detect and fix bugs in programs. In *ICLR*.

Ana Gonzalez, Isabelle Augenstein, and Anders Søgaard. 2018. A strong baseline for question relevancy ranking. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4810–4815.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014a. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014b. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

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

Daya Guo, Duyu Tang, Nan Duan, Ming Zhou, and Jian Yin. 2019. Coupling retrieval and meta-learning for context-dependent semantic parsing. In *ACL*.

Sonia Haiduc, Gabriele Bavota, Andrian Marcus, Rocco Oliveto, Andrea De Lucia, and Tim Menzies. 2013. Automatic query reformulations for text retrieval in software engineering. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 842–851. IEEE Press.

Tatsunori B. Hashimoto, Kelvin Guu, Yonatan Oren, and Percy Liang. 2018. A retrieve-and-edit framework for predicting structured outputs. *ArXiv*, abs/1812.01194.

Shirley Anugrah Hayati, Raphael Olivier, Pravalika Avvaru, Pengcheng Yin, Anthony Tomasic, and Graham Neubig. 2018. Retrieval-based neural code generation. In *EMNLP*.

Emily Hill, Manuel Roldan-Vega, Jerry Alan Fails, and Greg Mallet. 2014. Nl-based query refinement and contextualized code search results: A user study. In *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pages 34–43. IEEE.

Po-Sen Huang, Chenglong Wang, Rishabh Singh, Wen tau Yih, and Xiaodong He. 2018. Natural language to structured query generation via meta-learning. In *NAACL-HLT*.

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 (Volume 1: Long Papers)*, pages 2073–2083.

Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446.

Dong-Jin Kim, Jinsoo Choi, Tae-Hyun Oh, and In So Kweon. 2019. Image captioning with very scarce supervised data: Adversarial semi-supervised learning approach. In *EMNLP/IJCNLP*.

Alexander LeClair and Collin McMillan. 2019. Recommendations for datasets for source code summarization. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3931–3937, Minneapolis, Minnesota. Association for Computational Linguistics.

Dianqi Li, Qiuyuan Huang, Xiaodong He, Lei Zhang, and Ming-Ting Sun. 2018. Generating diverse and accurate visual captions by comparative adversarial learning. *ArXiv*, abs/1804.00861.

Shangsong Liang. 2019. Unsupervised semantic generative adversarial networks for expert retrieval. In *The World Wide Web Conference*, pages 1039–1050. ACM.

Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2017. Adversarial multi-task learning for text classification. In *ACL*.

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

Jing Ma, Wei Gao, and Kam-Fai Wong. 2019. Detect rumors on twitter by promoting information campaigns with generative adversarial learning. In *The World Wide Web Conference*, pages 3049–3055. ACM.

Mehdi Mirza and Simon Osindero. 2014. Conditional generative adversarial nets. *ArXiv*, abs/1411.1784.

Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. 2015. Distributional smoothing with virtual adversarial training. *arXiv preprint arXiv:1507.00677*.

Jong-Hoon Oh, Kazuma Kadowaki, Julien Kloetzer, Ryu Iida, and Kentaro Torisawa. 2019. Open-domain why-question answering with adversarial learning to encode answer texts. In *ACL*.

Sheena Panthaplackel, Milos Gligoric, Raymond J. Mooney, and Junyi Jessy Li. 2019. Associating natural language comment and source code entities. *ArXiv*, abs/1912.06728.

Sheena Panthaplackel, Pengyu Nie, Milos Gligoric, Junyi Jessy Li, and Raymond J. Mooney. 2020. Learning to update natural language comments based on code changes. *ArXiv*, abs/2004.12169.

Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255, Austin, Texas. Association for Computational Linguistics.

Dae Hoon Park and Yi Chang. 2019. Adversarial sampling and training for semi-supervised information retrieval. In *The World Wide Web Conference*, pages 1443–1453. ACM.

Pengda Qin, Weiran Xu, and William Yang Wang. 2018. Dsgan: Generative adversarial training for distant supervision relation extraction. In *ACL*.

Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. 2017. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7167–7176.

Venkatesh Vinayakarao, Anita Sarma, Rahul Purandare, Shuktika Jain, and Saumya Jain. 2017. Anne: Improving source code search using entity retrieval approach. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 211–220. ACM.

Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 515–524. ACM.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Yi Wu, David Bamman, and Stuart J. Russell. 2017. Adversarial training for relation extraction. In *EMNLP*.

Jiacheng Xu and Greg Durrett. 2018. Spherical latent spaces for stable variational autoencoders. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Xiao Yang, Madian Khabsa, Miaosen Wang, Wei Wang, Ahmed Hassan Awadallah, Daniel Kifer, and C Lee Giles. 2019. Adversarial training for community question answer selection based on multi-scale matching. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 395–402.

Ziyu Yao, Jayavardhan Reddy Peddamail, and Huan Sun. 2019. Coacor: Code annotation for code retrieval with reinforcement learning. In *The World Wide Web Conference*, pages 2203–2214. ACM.

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. International World Wide Web Conferences Steering Committee.

Wei Ye, Rui Xie, Jinglei Zhang, Tianxiang Hu, Xiaoyin Wang, and Shikun Zhang. 2020. Leveraging code generation to improve code retrieval and summarization via dual learning. In *Proceedings of The Web Conference 2020*, pages 2309–2319.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*.