Disentangle Identity and Semantic Information in Code Search

Sheng Zhang 23020221154146
Min-Hua Zheng 23020221154151
Yu-Bin Zhang 23020221154149
Information Class

Abstract

Code search is a foundational task in software development, which studies semantic similarity between natural language queries and program code. Recent years have witnessed great progress made in code search. However, researchers tend to pre-train a code representation model in different program languages so that it can be used in code related downstream problems and then fine-tune it in code search with a specific program language. Our empirical study shows that there may be performance damage when fine-tuning code search in multiple program languages. We believe that it is caused by the entanglement between code identity information and code sentiment information. Therefore, we proposed two disentangling strategies. One is leveraging a generator to obtain vectors without identity information, based on the idea of GAN (Generative Adversarial Network). The other is to maximize the KL (Kullback–Leibler) divergence between identity and semantic vectors.

1 Introduction

Code search plays a vital role in the software development process, which is an essential field of Software Engineering and studies the semantic similarity between natural language queries and program code. Recent years have witnessed a massive increment in source code. The statistic shows that more than 60 million new projects were created only in 2020 (Forsgren et al. 2021). Thus, code search engines can improve the development efficiency of program developers, enabling them to search for existing code or examples of some API (Application Programming Interface) instead of "rebuilding wheels."

As deep learning has grown by leaps and bounds in recent years, a number of methods have been proposed in code search, such as Recurrent Neural Network (RNN) based models (Gu, Zhang, and Kim 2018), CNNs (Convolutional Neural Networks) based models (Li et al. 2020; Shuai et al. 2020), graph based models (Gu, Chen, and Monperrus 2021), and Pre-trained Language Models (PLMs) based models (Feng et al. 2020; Huang et al. 2021; Guo et al. 2021, 2022).

From the view of PLMs, all of them have well performance in code search, with complex model architecture and advanced training techniques. However, PLMs often treat code search as a downstream task, which means researchers can pre-train a model with hybrid objectives and multiple program language code data, then fine-tune it in a specific program language for code search (Guo et al. 2022; Feng et al. 2020; Guo et al. 2021; Niu et al. 2022). Our empirical study shows that there might be a performance decline when fine-tuning with data in multiple languages. Table 1 shows MRR (Mean Reciprocal Rank, Voorhees 1999) comparisons between single program language fine-tuned models, and multiple program language fine-tuned models. The left first column indicates what program languages have been used in fine-tuning and the rest columns show the search performance toward a specific language according to the first column. Similar to multilingual models (Yu, Fei, and Li 2021; Yang et al. 2021), code information can be roughly divided into identity information, distinguishing code from another code written in different program languages, and semantic information, which reveals its intention and with a corresponding natural language description. In code search, code semantic information is only needed, for it is matched with specific queries. The identity information may confound model training and decrease performance when fine-tuning with multiple program language data. We view identity information as the signal that helps to distinguish different program language data, which has the same idea as classification. Semantic information reveals the code intention, which is related to code search.

Therefore, we want to reduce the identity information of the embedding vector given by code search pre-trained models. We propose two strategies for disentangling. The first is to follow the idea of GAN (Generative Adversarial Network, Goodfellow et al. 2020) and leverage a generator to generate identity free embedding vectors. The second is to use an additional network to obtain identity and semantic vectors separately. We consider maximizing KL divergence in the loss function.

In summary, our contributions are:

1. Reveal the performance decline problem that appears when fine-tuning code search with multiple program languages.
2. We propose different disentangling strategies for splitting identity and semantic information of the embedding vectors output by pre-train code models.

Table 1: Performance comparisons with different fine-tuning program languages

	Python	Go	Java	PHP
Python	0.369	_	_	
Go	_	0.904	_	_
Java	_	_	0.764	_
PHP	_	_	_	0.708
Python_Java	0.468	_	0.708 ↓	_
Python_Go	0.404	0.882 ↓	_	_
Java_Go	_	0.902 ↓	0.724 ↓	_
Java_PHP	_	_	0.743 ↓	0.707 ↓
Java_Python_Go	0.571	0.889 ↓	0.701 ↓	_

2 Related Work

Our work is related to the following two fields:

2.1 Code Search

Recent years' works adopt deep learning models in code search, of which the idea is to embed natural language queries and program code into vectors and then calculate their similarity score. Fig 1 shows the framework of code search: a code encoder, a natural language encoder, and a similarity calculator. The encoder transforms code or queries into high-dimensional vectors, which are considered semantic information. The calculator o computes the similarity between the two vectors.

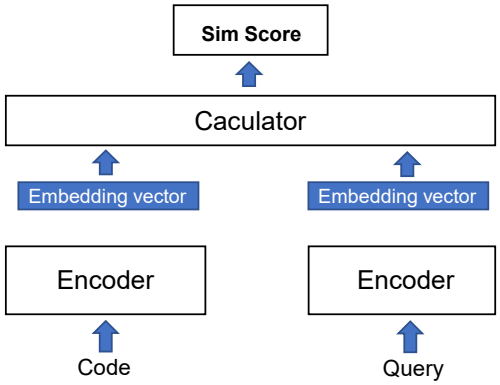

Figure 1: Structure of the search enhancement framework.

Models in code search based on deep learning can be roughly divided into the following aspects:

(1) RNN based models. Gu et al. (Gu, Zhang, and Kim 2018) build two networks to embed queries and program code into vectors, respectively. Cosine similarity is used to compute the similarity between the vectors.

(2) CNNs based models. Followed by (Gu, Zhang, and Kim 2018), Li et al. (Li et al. 2020) construct "name-query" and "body-query" latex match matrix with fastText (Joulin et al. 2016) and use CNNs to extract features.

(3) PLMs. Feng et al. (Feng et al. 2020) followed the idea of BERT (Devlin et al. 2019) and proposed a pre-train model training on CSN (Husain et al. 2019) dataset with hybrid objective functions: MLM (Masked Language Modeling) and RTD (Replaced Token Detection). (Guo et al. 2021) leverage data flow to enhance code representation. For the code data flow, they specifically designed a series of pre-training tasks: MLM, Edge Prediction, and Node Alignment. (Guo et al. 2022) tranfrom code AST (Abstract Syntex Tree) to a sequence structure, thus the pre-training can utilize multimodal contents.

2.2 Generative Adversarial Nets

Adversarial training is an effective approach to enhance the robustness of a deep learning model(Bai et al. 2021). And generative models are able to learn the probability distribution that generated the training examples. Generative adversarial networks(GANs) then performs well in generating more examples from the estimated probability distribution(Goodfellow et al. 2020) so that it becomes a hot research topic recently.

Generally, GAN(Goodfellow et al. 2014) is similar to a minimax two-player game: one is a generative model G that captures the data distribution, and the other player is a discriminative model D that estimates the probability that an example came from the training data rather than G, i.e. to determine whether an example is from the model distribution or the data distribution. Both G and D are defined by multilayer perceptrons and are trained simultaneously. Competitions among these two players drives both to improve their performance respectively. And the value function in this game can be referred to as

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[log(1-D(G(z)))] \quad (1)$$

where $p_{data}(x)$ is the distribution over data x, z is the input noise, $G(z)$ represents a mapping from z to data space, and $D(x)$ refers to the probability that x came from the data rather than G.

In practice, the GANs must be implemented using an iterative, numerical approach, i.e. alternating between k steps of optimizing D and one step of optimizing G. And finally the training or the game terminates at a saddle point that is a minimum with respect to one player's strategy and a maximum with respect to the other player's strategy.

3 Methodology

We propose two strategies to improve code search performance based on the assumption that identity information will confuse the model when fine-tuning multiple language data. Instead of changing the structure and pre-training the whole code search model, we focus on transforming the model output embedding vectors during the fine-tuning process, which is less computing complexity.

The first strategy is leveraging GAN to eliminate identity information. The second strategy is to distance the KL divergence (Yu et al. 2013) between identity embedding and semantic embedding.

Identity and semantic information. The identity information we consider as the syntax signal of different program languages, which distinguishes them from each other. We

consider the identity information as the signal that the model can classify each program language data correctly. The semantic information is the intention of a code snippet, which describes the function of the code.

3.1 Strategy 1

In this section, we introduce a GAN-based enhance network, which contains a generator and a discriminator. We start by introducing the basic idea and the model structure. Then we describe the training procedure in this paper in detail.

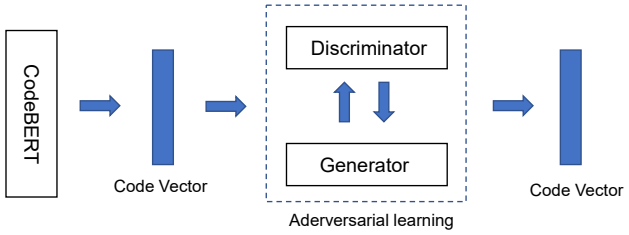

Figure 2: Structure of the search enhancement framework.

The GAN structure is used widely in image generation. Generally, it contains two sub-network: a generator, which aims at generating data of a specific distribution, and a discriminator, which intends to determine the generated data if it is in the ideal distribution. In this paper, we introduce an MLP (Multilayer Perceptron) as the generator, of which the purpose is to generate high-dimensional vectors free of identity information. We also use another MLP as the discriminator. The object of the discriminator is to determine whether the vector generator produced contains identity information. The structure of the network can be seen in Fig 2.

Algorithm 1: **Stragegy 1 training process**

```
for epoch_idx in total_epoch_num:
  for step, batch in dataloader:
    code_inputs, nl_inputs, labels =
        batch
    code_vec, nl_vec = CodeBERT(
        code_inputs, nl_inputs)
    search_embedding_vec = generator(
        code_vec)
    if step < 2000:
      train_search_model()
    elif (step / 500) % 2 == 0 and step
        >= 2000:
      train_discriminator()
    elif step >= 2000 and step / 500 % 2
        != 0:
      train_generator()
```

We give the training process of strategy 1 in Algorithm 1. First, we obtain the embedding vectors of code and natural language descriptions. At the first 2,000 step of a epoch, we update the code search process, with the object of minimizing the distance between code snippets and their corresponding descriptions. Then, alternating update generator and discriminator every 500 steps.

3.2 Strategy 2

Although GAN has a strong ability of learning and manipulating data distribution, it is hard to be trained, and can easily suffer from gradient vanishing (Adler and Lunz 2018). Therefore, we propose an MLP-based network to disentangle identity and semantic information. The overlook of the network can be seen in Fig 3.

The network takes embedding vectors extracted by CodeBERT as the input. Then, the input vector will be put into MLPs and can obtain two high-dimensional vectors: V_1 and V_2. We hope that V_1 possesses identity information and V_2 possesses semantic information. Therefore, we use V_1 for classification and V_2 for code search, in order to extract identity and semantic information, respectively. Then, we maximize KL divergence, which is illustrated in Eq, between V_1 and V_2. Finally, V_2 can be viewed as the sentence embedding of a input code snippet without identity information and can be further used for code search.

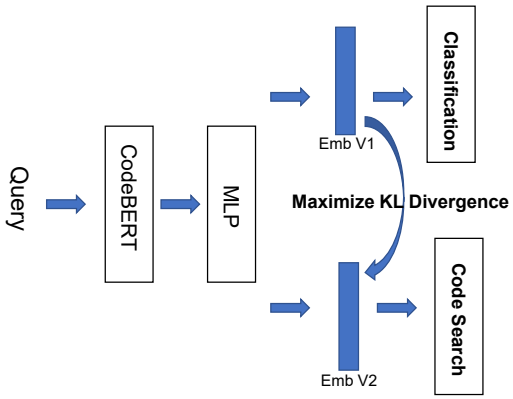

Figure 3: Overlook of strategy 2

According to the above, learning the disentangle network involves finding parameters $\theta = (W, b)$ that minimize the hybrid loss on the given dataset, and the objective is given as Eq 5, where $\alpha_1, \alpha_2, \alpha_3$ are hyperparameters, and the sum of them is equal to 1. Besides, Eq 2 represents the code search loss, intends to keep search performance, where sim is the similarity score and c_i is the i_{th} code and q_i is the i_{th} query. Eq 3 is the classification loss, the purpose is to extract identity information. Eq 4 leverages KL divergence to maximize the distribution of the embedding vectors output by the disentangle network.

$$L_1 = \frac{1}{n}\sum_{i=1}^{n}\left(1 - sim\left(encoder\left(c_i\right), encoder\left(q_i\right)\right)\right) \quad (2)$$

$$L_2 = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} -y_{i,j} log\left(p_{i,j}\right) \quad (3)$$

$$L_3 = \frac{1}{n}\sum_{i=1}^{n}\left(\alpha - KL\left(V_{1,i}, V_{2,i}\right)\right) \quad (4)$$

$$\theta = \min_{\theta} \alpha_1 L_1 + \alpha_2 L_2 + \alpha_3 L_3 \quad (5)$$

4 Experiment

4.1 Experiment Setting

Dataset. The dataset we used is the Code Search Net dataset (Husain et al. 2019) preprocessed by GraphCodeBERT (Guo et al. 2021). The parameter of the dataset is given below. We remove the data of Ruby and Javascript, as they are much less than other data.

Baseline model. We choose CodeBERT (Feng et al. 2020) as the baseline model. It plays an important role in extracting code and query features. It is the encoder that first embed a code snippet or a natural query into a high-dimensional vector.

Disentangle network. We use multiple linear perceptron as the basic layers of the network. We use ReLU (Agarap 2018) as the activation.

Training setting. We use an RTX 3090 GPU for training. The batch size of the training procedure is 128. The total training epoch number is 3. We use the package of Transformers 4.24.0 and Pytorch 1.12.0, based on Python 3.8.13.

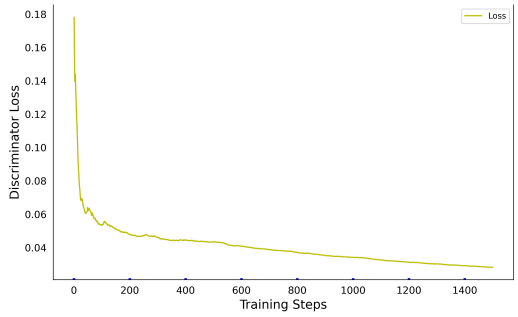

Figure 4: Loss change of discriminator during training.

4.2 Analysing Training Process

Disentangle Network 1. We illustrated the change of discriminator loss during training in Fig 4. It is obviously that the loss of the discriminator is able to converge, while there is still slightly increment in some period, e.g., step 50 and step 100. Fig 5 shows the loss change of the generator during training. There might be overfitting, as the loss first decreases and then rises with increasing iterations.

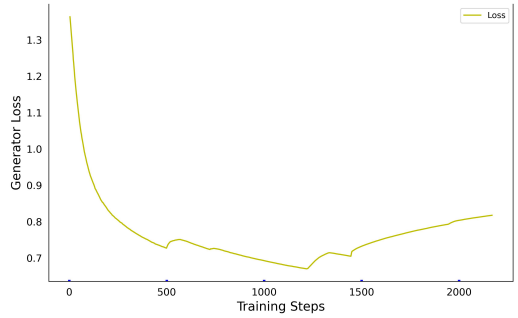

Figure 5: Loss change of generator during training.

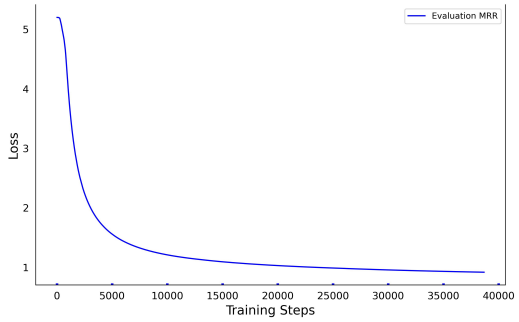

Figure 6: Loss of network 2 during training.

Disentangle Network 2. We list hybrid loss change of network based on strategy 2 in Fig 6. It is apparent the loss is decreasing and able to converge. Fig 7 represents the evaluation index change in the training period, using MRR, which is shown in Eq 6, as the index and with the evaluation step of 2, 000.

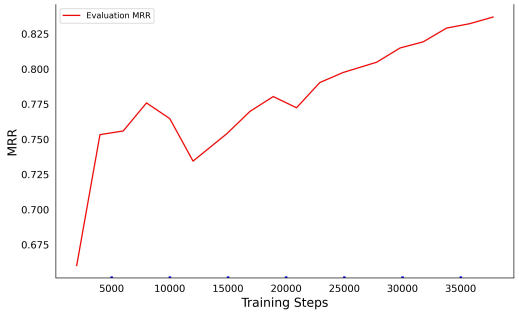

Figure 7: MRR change in evaluation during training.

4.3 Analysing Embedding Vectors after Disentangling

Intuitively, identity and semantic information can be divided easily after applying the disentangle strategy we proposed. Thus, we first use t-SNE (Van der Maaten and Hinton 2008), which is a dimensionality reduction method, to reduce high-dimensional vectors to 2 dimensions. Then we visualize the 2-dimensional vectors. We sample 5, 000 code snippets from the whole dataset. And then draw the embedding graph. For the strategy 1 network, we visualize the vectors output by the origin CodeBERT and the generator. For the strategy 1, we illustrate embedding vectors output by CodeBERT and disentangle network in Fig 9. Compared to the CodeBERT vectors, i.e., blue points, the generator output, i.e., yellow points, is less concentrated in the same area. For the strategy 2, we visualize v1 and v2 vector, which is considered have identity and semantic information, respectively. From the graph 8 we can see that, two types of the vectors can basically be separated.

4.4 Analysing Code Search Results

In this section, we try to figure out the code search performance of two disentangle networks. We use MRR as the

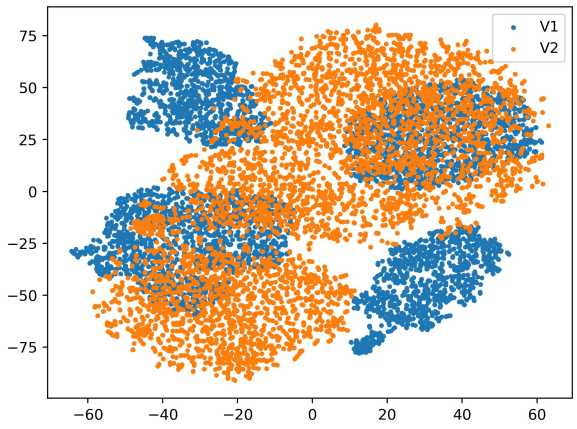

Figure 8: Embedding vector comparision of strategy 2 network.

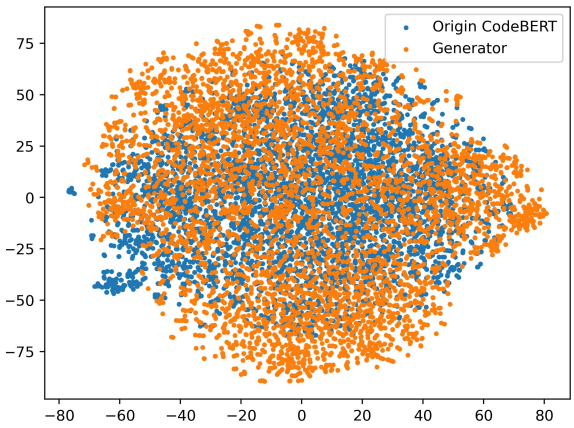

Figure 9: Embedding vector comparision of strategy 1 network.

evaluation indicator, which is the reciprocal of the correct result among all returned results and it is given as Eq 6, where Q is the total test number and $rank_i$ represents the correct rank among all results..

$$MRR = \frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{rank_i} \tag{6}$$

Table 2: MRR Comparisions

	Strategy 1	Strategy 2
Python	0.597	0.601
Go	0.873	0.882
Java	0.712	0.703
Php	0.684	0.673

Table 1 shows the code search MRR results. The left first column indicates the language of data. The top row indicates the strategy we proposed. From the results we can see that there is a small performance increment in Python, compared to the original results in 1, while a slightly decrease occurs in other languages. It is clear that strategy 1 and strategy 2 can reach a similar result, while Java and Php data have a better MRR in strategy 1, and Python and Go data are better in strategy 2.

5 Discussion and Conclusion

In this paper, we propose two disentangle strategies for tackling performance decline when fine-tuning CodeBERT for the code search task with multiple language data. In strategy 1, we leverage a GAN-based model, to eliminate identity information. In strategy 2, we use MLPs to divide embeddings by maximize their KL divergence. Experiments show our work achieve some results.

However, we only consider identity information as the signal that distinguish program languages from each other, e.g., a classification task. Whether the identity information is equal to syntax information, needs more experiments to proof. Besides, the code search performance still have room for improvement. We hope our work can be reference for the same field researchers.

References

Adler, J.; and Lunz, S. 2018. Banach wasserstein gan. *Advances in neural information processing systems*, 31.

Agarap, A. F. 2018. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*.

Bai, T.; Luo, J.; Zhao, J.; Wen, B.; and Wang, Q. 2021. Recent advances in adversarial training for adversarial robustness. *arXiv preprint arXiv:2102.01356*.

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT (1)*, 4171–4186.

Feng, Z.; Guo, D.; Tang, D.; Duan, N.; Feng, X.; Gong, M.; Shou, L.; Qin, B.; Liu, T.; Jiang, D.; and Zhou, M. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *EMNLP (Findings)*, 1536–1547.

Forsgren, N.; Alberts, B.; Backhouse, K.; Baker, G.; Cecarelli, G.; Jedamski, D.; Kelly, S.; and Sullivan, C. 2021. 2020 State of the Octoverse: Securing the World's Software. *CoRR*, abs/2110.10246.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2020. Generative adversarial networks. *Communications of the ACM*, 63(11): 139–144.

Goodfellow, I. J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative Adversarial Nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, 2672–2680. Cambridge, MA, USA: MIT Press.

Gu, J.; Chen, Z.; and Monperrus, M. 2021. Multimodal Representation for Neural Code Search. In *ICSME*, 483–494.

Gu, X.; Zhang, H.; and Kim, S. 2018. Deep code search. In *ICSE*, 933–944.

Guo, D.; Lu, S.; Duan, N.; Wang, Y.; Zhou, M.; and Yin, J. 2022. UniXcoder: Unified Cross-Modal Pre-training for Code Representation. In *ACL (1)*, 7212–7225.

Guo, D.; Ren, S.; Lu, S.; Feng, Z.; Tang, D.; Liu, S.; Zhou, L.; Duan, N.; Svyatkovskiy, A.; Fu, S.; Tufano, M.; Deng, S. K.; Clement, C. B.; Drain, D.; Sundaresan, N.; Yin, J.; Jiang, D.; and Zhou, M. 2021. GraphCodeBERT: Pre-training Code Representations with Data Flow. In *ICLR*.

Huang, J.; Tang, D.; Shou, L.; Gong, M.; Xu, K.; Jiang, D.; Zhou, M.; and Duan, N. 2021. CoSQA: 20, 000+ Web Queries for Code Search and Question Answering. In *ACL/IJCNLP (1)*, 5690–5700.

Husain, H.; Wu, H.; Gazit, T.; Allamanis, M.; and Brockschmidt, M. 2019. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. *arXiv Preprint*.

Joulin, A.; Grave, E.; Bojanowski, P.; Douze, M.; Jégou, H.; and Mikolov, T. 2016. FastText.zip: Compressing text classification models. *CoRR*, abs/1612.03651.

Li, W.; Qin, H.; Yan, S.; Shen, B.; and Chen, Y. 2020. Learning Code-Query Interaction for Enhancing Code Searches. In *ICSME*, 115–126.

Niu, C.; Li, C.; Ng, V.; Ge, J.; Huang, L.; and Luo, B. 2022. SPT-Code: Sequence-to-Sequence Pre-Training for Learning Source Code Representations. In *ICSE*, 1–13. ACM.

Shuai, J.; Xu, L.; Liu, C.; Yan, M.; Xia, X.; and Lei, Y. 2020. Improving Code Search with Co-Attentive Representation Learning. In *ICPC*, 196–207.

Van der Maaten, L.; and Hinton, G. 2008. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11).

Voorhees, E. M. 1999. The TREC-8 Question Answering Track Report. In *TREC*, volume 500-246 of *NIST Special Publication*. National Institute of Standards and Technology (NIST).

Yang, Z.; Yang, Y.; Cer, D.; and Darve, E. 2021. A Simple and Effective Method To Eliminate the Self Language Bias in Multilingual Representations. In *EMNLP (1)*, 5825–5832. Association for Computational Linguistics.

Yu, D.; Yao, K.; Su, H.; Li, G.; and Seide, F. 2013. KL-divergence regularized deep neural network adaptation for improved large vocabulary speech recognition. In *ICASSP*, 7893–7897. IEEE.

Yu, P.; Fei, H.; and Li, P. 2021. Cross-lingual Language Model Pretraining for Retrieval. In *WWW*, 1029–1039.