Improving Programming Q&A with Neural Generative Augmentation

Suthee Chaidaroon, Xiao Zhang, Shruti Subramaniyam, Jeffrey Svajlenko,
Tanya Shourya, Iman Keivanloo, Ria Joy
Amazon Web Service
USA
{sutheec,zhxao,sushruti,jesvajle,tshourya,imankei,regjoy}@amazon.com

ABSTRACT

Knowledge-intensive programming Q&A is an active research area in industry. Its application boosts developer productivity by aiding developers in quickly finding programming answers from the vast amount of information on the Internet. In this study, we propose **ProQANS** and its variants **ReProQANS** and **ReAugProQANS** to tackle programming Q&A. **ProQANS** is a neural search approach that leverages unlabeled data on the Internet (such as StackOverflow) to mitigate the cold-start problem. **ReProQANS** extends **ProQANS** by utilizing reformulated queries with a novel triplet loss. We further use an auxiliary generative model to augment the training queries, and design a novel dual triplet loss function to adapt these generated queries, to build another variant of **ReProQANS** termed as **ReAugProQANS**. In our empirical experiments, we show **ReProQANS** has the best performance when evaluated on the in-domain test set, while **ReAugProQANS** has the superior performance on the out-of-domain real programming questions, by outperforming the state-of-the-art model by up to 477% lift on the MRR metric respectively. The results suggest their robustness to previously unseen questions and its wide application to real programming questions.

ACM Reference Format:
Suthee Chaidaroon, Xiao Zhang, Shruti Subramaniyam, Jeffrey Svajlenko,, Tanya Shourya, Iman Keivanloo, Ria Joy. 2023. Improving Programming Q&A with Neural Generative Augmentation. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '23), July 23–27, 2023, Taipei, Taiwan.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3539618.3591860

1 INTRODUCTION

It is widely known that developing and maintaining software systems is a non-trivial task. A study by Brandt et al. [3] suggests developers spend 19% of their time searching for answers to their programming questions, while Xia et al. [21] found that professional developers spend 58% of their time on program comprehension tasks, which includes searching related online resources while in the development cycle. To facilitate the development speed, developers commonly search online to find answers to their programming questions during implementation and refactoring. In Figure 1, we show an example of a well-designed system able to answer a programming question.

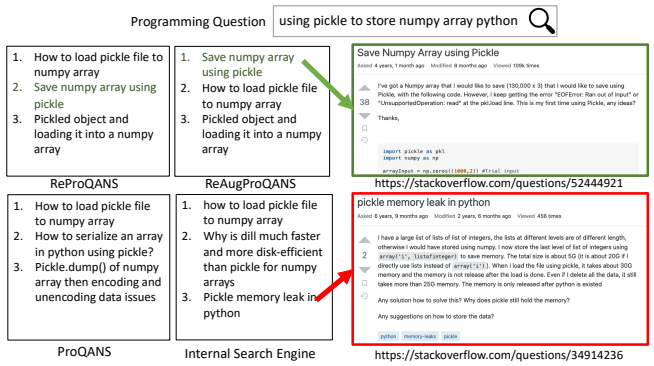

Figure 1: An example of Knowledge-intensive Programming Q&A task. In this task, the system answers programming questions by locating relevant public content on the Internet.

Programming question and answering (Q&A) is usually cast as a retrieval problem, in which questions are regarded as queries and answers as documents. However, matured document retrieval systems cannot be directly applied to programming Q&A search [12]. Differing from traditional documents, these answers contain a mixture of natural and programming languages (PLs). In contrast to natural languages, PLs have a strict syntax with limited semantic cues, an unlimited vocabulary as developers create arbitrary variable, function and class names, with references to classes or functions always defined somewhere else.

Researchers have made efforts to tackle these challenges from two major complementary perspectives. Some developed techniques to reformulate the queries to enrich them with more information relevant to the related answer, by injecting, rewriting and expanding words in the queries [6, 7, 11]. Others propose to enhance the retrieval model with neural architectures to improve the model capacity and the internal representations [5, 18]. But a few challenges still remain open. Sridhara et al. [19] has found expanding the queries inappropriately may produce even worse results than not, while deep neural network based models are notoriously known to be susceptible to over-fitting with large number of parameters, without sufficient amount of training data. As a cold start issue, the lack of high quality data in the domain of programming Q&A is especially challenging for successfully training a neural search model.

In this work, we explore to improve Programming Q&A systems by enhancing the underlying neural search models without labeled

SIGIR '23, July 23–27, 2023, Taipei, Taiwan

ACM ISBN 978-1-4503-9408-6/23/07.
https://doi.org/10.1145/3539618.3591860

data, using weakly supervision instead. To this end, we show that by reforming the title and content of programming question posts as the question and answer for data augmentation, and use them together with query reformulation to train neural search models significantly improves programming Q&A performance. Additionally, we show using auxiliary generative models for data augmentation effectively improves the robustness of the model. Figure 2 illustrates the schema of this. We summarize our contributions as follows:

(1) We present a neural search model for programming Q&A by sampling both in-batch and global negatives, which is termed as **ProQANS**, and propose its variant **ReProQANS** that leverages the query reformulation information with a novel triplet loss.
(2) We further propose another variant referred to as **ReAugProQANS** by a data augmentation approach with an auxiliary underlying generative model to increase both the size and the quality of the training set, with a novel dual triplet loss.
(3) We show empirically **ReProQANS** achieves the best performance when evaluated on the in-domain data, while **ReAugProQANS** is robust on the real out-of-domain questions for programming Q&A. The proposed models outperform the state-of-the-art model by up to 477% lift on the MRR metric respectively.

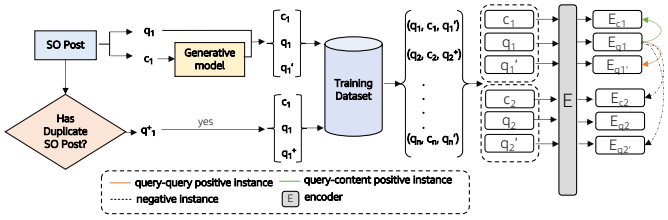

Figure 2: This diagram illustrates the process for creating training samplers. For each post, we check for the presence of another similar post by examining a duplicated tag provided by the SO dataset. If a similar post exists, we use its title as an alternate query. If no similar post is found, we employ a generative model to produce a new post title using the content of the original post. The resulting training examples comprises of the original title, the alternate title, and the content of the original post.

2 RELATED WORK

Early work on source code search focuses on utilizing the lexical information, e.g, identification of the importance of the positional information of the query words [7], examination of the context of words in comments and code [11], and augmentation of the API to be searched by documentation [14]. More recently, researchers explored source code retrieval, generation, summarizing and clone detection [1, 5, 9, 18, 23].

Data augmentation has been shown to be effective in neural search models. Nogueira et al. [15, 16] expanded the documents by using generative models to generate related queries and concatenating them to the corresponding documents to form new documents for training the retrieval model. Jain et al. [10] augmented their source code documents to be searched with code transformation. More recent studies include predicting the reason behind query reformulation [4], rewriting jargon words to formal expressions in the query using knowledge base [13], and building large-scale query reformulation data set using transformer architectures [2]. Cao et al. [4] conducted a study in which they utilized a transformer to reformulate and enhance search queries, which is the most closely related work to our research.

However, none of these works take into account the use of query reformulation to provide additional signal in the loss function when training the retrieval model. In this study, we explore to improve neural search for programming Q&A by leveraging query reformulation and training data augmentation, with newly designed loss functions.

3 METHODOLOGY

Open domain knowledge-intensive Q&A relies on a search model which retrieves relevant answers to a given question from a corpus of documents. In this setup, we treat the question and answer as query and document pairs, and train the neural search model on them. After training, the neural search model is expected to produce a higher score for relevant question and answer pairs.

3.1 Neural Search Model

In this study, we extend the neural search model used in Guo et al. [5] for programming Q&A. The neural model includes a single encoder consisting of a stack of transformer layers and an embedding head, which encodes the query or document to be a dense vector. In Figure 2, we outlines the architecture of the neural search model.

The neural search model is optimized via a contrastive loss during the training as following:

$$\mathcal{L}_{\text{ret}} = -\log\left(\frac{\exp(\frac{1}{\tau}\cdot f(q,d^+))}{\exp(\frac{1}{\tau}\cdot f(q,d^+)) + \sum_{d^-\in\mathcal{N}^-}\exp(\frac{1}{\tau}\cdot f(q,d^-))}\right), \tag{1}$$

where we denote q as the programming question (aka query) embedding, d^+ as a relevant document embedding generated via the embedding head aforementioned, while d^- as an irrelevant document embedding. The scoring function f computes a cosine similarity between query and document. τ is a hyperparameter that controls the degree of smoothing or sharpening in the output probability distribution from the softmax function. Differing from the model in Guo et al. [5]'s, we randomly sample a few irrelevant documents d^- from a noise distribution $\mathcal{N}^-$, which is a combination of in-batch and global negatives. This strategy is based both on our pilot experimental results and the discovery in Xiong et al. [22]'s study. We term this neural search model as **ProQANS** (**Pro**gramming **Q**&**A** **N**eural **S**earcher).

3.2 Modeling Reformulated Queries

Different developers may use different phrasing to describe the same programming questions. By exposing the model to multiple variations of the same programming question, the underlying model can learn to respond consistently, regardless of the specific wording used.

To this end, we design a novel loss that consumes query reformulation during contrastive learning, which is a triplet loss to encourage the model to produce higher similarity score between reformulated

queries. The loss function is defined as follows:

$$\mathcal{L}_{\text{q}}(q, q^+, q^-) = \max(f(q, q^+) - f(q, q^-) + \xi, 0), \quad (2)$$

where q acts as an anchor query embedding, and q^+ is a related query embedding, with q^- as an unrelated query embedding. The model is forced to maximize the similarity score between q and q^+ while minimize the score between q and q^-. We sample q^- uniformly from the training corpus and set the margin ξ to 1.0. We term this model variant as **ReProQANS** (**Re**formulation **ProQANS**).

3.3 Modeling Data Augmentation

Gathering a comprehensive set of variations for programming questions poses a challenge, even with the use of sources such as Stack Overflow duplicated dataset. As Table 1 highlights, only a small portion, between 2-8%, of the queries have multiple variations. With the aim of increasing the quantity of query reformulations, we have integrated an auxiliary generative model into our model training framework.

We employ a state-of-the-art generative model to produce supplementary queries for each post. The auxiliary model is trained on the text of a Stack Overflow post after the original title has been removed, and its task is to generate a title that is similar to the original. We outline the process of query augmentation via generation in Figure 2. Once the query generation step is complete, we create additional training data containing pairs of original and generated queries. The new loss function has two triplet losses:

$$\mathcal{L}_{\text{qp}} = \delta(q) \cdot \mathcal{L}_{\text{q}}(q, q^+, q^-) + (1 - \delta(q))(\mathcal{L}_{\text{q}}(q, \tilde{q}, q^-)) \quad (3)$$

The first term is the query prediction loss between duplicated q^+ and relevant query embeddings q, described in Eq.2. The second term considers the generated query $\tilde{q}$ as a positive query and computes a max-margin loss that compares it with the original query q and negative sampling query q^-, similar to the first term. The Indicator function $\delta(q)$ is evaluated to 1 if the given post is marked as duplicated, otherwise it is set to 0.

We train the model using the loss from Eq.1 and Eq.3. Thus, the loss for a single sample is $\mathcal{L} = \mathcal{L}_{\text{ret}} + \omega \cdot \mathcal{L}_{\text{qp}}$, where the tunable parameter ω controls the influence of the query prediction loss, $\mathcal{L}_{\text{qp}}$. The first loss enforces the query and document embeddings to be as close as possible and the second loss encourages the model to group similar queries together; therefore, not only the query embeddings but also the document embeddings of similar queries likely to be close to each other. We term this variant as **ReAugProQANS** (**Re**formulation **Aug**mentation **ProQANS**).

4 EXPERIMENTS

4.1 Data Set

Stack Overflow (SO) Data. We process the publicly available SO data dumped on June, 2022[1][20] for creating the training set that fits our contrastive learning requirements. We first filter the posts related to three programming languages: *python*, *javascript* and *typescript*, to create 3 PL data set. For each of them, we treat the title of each post as the query and the rest as the content of the document to automatically create query-document pairs at scale. We randomly split all the queries into training, validation, and testing sets with the ratio of 8:1:1. We further include the titles from duplicative posts and generated queries as query reformulations.

[1] https://archive.org/download/stackexchange/stackoverflow.com-Posts.7z

Data sets	Queries			Documents	Duplicates
	#Train	#Dev	#Test	#Doc	#Dup
python	1905280	238160	238160	2381600	177152
javascript	1568160	196020	196020	1960200	166626
typescript	147333	18416	18416	184165	4256

Table 1: The statistics of the Stack Overflow data set, including the numbers of queries divided as training, validation and testing set, and the number of documents as the corpus. We also include the statistics of duplicate posts using the signal from SO.

Extended CoSQA. CoSQA [8] is a dataset containing 20,604 programming questions and code snippet pairs. The programming questions in CoSQA are extracted from the search log of Microsoft's Bing search engine. We extend CoSQA to support programming Q&A by matching the questions in the original data set to a relevant answer from the SO forum, to form question (query)-answer (document) pairs. We use this data set for evaluation only.

User PL QA Queries (PLQA). A internal search engine is designed to support software developers to find relevant answers to their questions. We utilize the historical search log from it that are programming questions formed by real users to create a set of representative queries. We manually reviewed each candidate question in the log to discard irrelevant items and sift a number of high quality ones. After data cleaning, we randomly selected a set of 88 *python* programming questions. We then identify one relevant answer from SO for each programming question in this set as the relevant document. We use this data set for evaluation only.

4.2 Experimental Setup

We compare our proposed models with the state-of-the-art model which have been extensively used for source code retrieval: UniXcoder [5]. Our models are initialized with the encoder weights from the pretrained UniXcoder model. In **ReAugProQANS**, we adopt the T5 model [17] as the auxiliary generative model and fine-tune it on the training set to generate queries to form the augmented data. Throughout our evaluation, we report both recall and Mean Reciprocal Rank (MRR) as retrieval performance metrics.

4.3 Out-of-domain Evaluation

Real PL questions are different from SO titles, with the intent behind the developers to resolve the issues related to the source code they are developing. These real queries are usually less rigorously formed, with acronyms as unusual words, or only related to a specific aspect of the code without a general audience. These queries are usually from a different distribution than the queries we use for training our neural retrieval models. To answer the research question whether the learning of the trained models can be transferred to a different distribution of real programming questions, we design the experiments of evaluating our models by using the Extended CoSQA data set and User PL QA Queries data set as out-of-domain test sets.

Evaluation on Extended CoSQA. Our evaluation of the proposed models on the Extended CosQA out-of-domain dataset is presented

in Table 2. The results clearly demonstrate that our ProQANS and ReProQANS models outperform the current state-of-the-art UniXcoder baselines. In addition, **ReAugProQANS** achieves the best performance among all the models in out-of-domain evaluation, implying its ability to adapt to queries from a different domain. We attribute this success to the underlying generative augmentation module we use to generate the augmented questions (queries). The generated queries used as training examples expand the distribution of queries the model can access during the training and prevent the model from over-fitting to the original query distribution. The robustness of our approach is particularly useful for new PL questions as queries from a different domain.

Evaluation on User PL QA Queries. We further evaluate our approach on the User PL QA Queries, which are real user queries we collect from the internal search engine. In this evaluation, we observe the same tendency as the evaluation on the Extended CoSQA set: **ProQANS** and **ReProQANS** outperform UniXcoder and Lexical matching baselines, and **ReAugProQANS** performs best among all the models. These results echo and validate our findings in the evaluation of the Extended CoSQA set, that **ReAugProQANS** is a robust method when applied to out-of-domain queries from training, less prone to over-fitting, and is able to benefit the developers as it handles real user queries from a new domain.

Benchmark	Model	MRR	R@1	R@5	R@10	R@20	R@100
CoSQA	Lexical Matching	0%	0%	0%	0%	0%	0%
	UniXcoder	59.59%	70.98%	47.76%	48.41%	52.33%	150.76%
	ProQANS	131.10%	142.74%	127.12%	121.08%	123.18%	247.77%
	ReProQANS	152.76%	168.87%	146.12%	140.01%	142.31%	267.56%
	ReAugProQANS	**167.01%**	**187.60%**	**156.44%**	**151.07%**	**152.68%**	**272.77%**
PLQA	Lexical Matching	0%	0%	0%	0%	0%	0%
	UniXcoder	104.16%	195.80%	87.61%	30.75%	16.66%	83.34%
	ProQANS	129.46%	195.80%	125.11%	61.50%	74.01%	172.19%
	ReProQANS	137.78%	200.00%	125.11%	84.56%	77.79%	194.45%
	ReAugProQANS	**141.59%**	**200.00%**	**162.61%**	**99.94%**	**83.34%**	**205.55%**

Table 2: The performance on testing different models on the out-of-domain python queries. ReAugProQANS achieves the best performance due to its robustness to queries from a shifted distribution. The percentage listed in the table indicates the relative improvement over the Lexical matching baseline.

4.4 In-domain Evaluation

We show the results of different models on the in-domain SO test set in Table 3. Our proposed models outperform the strong UniXcoder baselines across all the programming languages on both the MRR and the Recall metrics by a large margin. We attribute this success to the additional fine-tuning and the specifically designed loss functions adapted to the Q&A scenario. When comparing the results among the proposed models, we notice **ReProQANS** has the best performance on the in-domain test set, beating **ReAugProQANS**. We hypothesize this is due to the augmented training set diverges from the original training set, thus leading to degraded performance on the in-domain testing set.

5 CONCLUSION

In this study, we propose our neural search model **ProQANS** to answer PL questions by searching over the SO posts, and its variants

Language	Model	MRR	R@1	R@5	R@10	R@20	R@100
python	UniXcoder	0%	0%	0%	0%	0%	0%
	ProQANS	334.25%	399.32%	293.57%	251.91%	217.79%	150.30%
	ReProQANS	**379.36%**	**464.22%**	**323.84%**	**276.45%**	**235.63%**	**159.63%**
	ReAugProQANS	303.22%	331.86%	264.58%	235.79%	206.48%	150.00%
Typescript	UniXcoder	0%	0%	0%	0%	0%	0%
	ProQANS	271.90%	326.61%	234.22%	**195.56%**	168.89%	**112.85%**
	ReProQANS	**274.36%**	**329.31%**	**236.99%**	193.54%	**170.27%**	112.59%
	ReAugProQANS	251.81%	298.34%	217.34%	184.84%	165.13%	110.72%
Javascript	UniXcoder	0%	0%	0%	0%	0%	0%
	ProQANS	431.15%	515.09%	377.13%	317.63%	272.32%	173.09%
	ReProQANS	**476.78%**	**583.95%**	**406.93%**	**338.85%**	**285.89%**	**184.43%**
	ReAugProQANS	366.39%	412.77%	339.11%	293.13%	253.85%	177.59%

Table 3: The relative percent increase in performance across various models on the in-domain SO test set. ReProQANS achieves the best performance, and all of our proposed models outperform the strong UniXcoder baseline. We exclude the performance of lexical matching due to its poor performance.

ReProQANS and **ReAugProQANS**. **ProQANS** adapts the neural search architecture with contrastive learning to our PL question answering search, with both in-batch and globally sampled negatives, while **ReProQANS** leverages the query reformulation information to enhance **ProQANS** with a triplet loss, encouraging the model to learn similar query representations between similar queries. **ReProQANS** achieves the best performance when being evaluated on the in-domain test set. In addition, we propose **ReAugProQANS**, using an underlying generative model to augment the queries in the training set, with a novelly designed dual triplets loss, to utilize the augmented data for training. While being evaluated on the out-of-domain query test set that are genuine PL questions from real developer, **ReAugProQANS** has the superior performance over all other competitors, suggesting its robustness to the shift of query distribution when searching for appropriate answers among SO posts to real programming questions. Its ability to be adapted to unseen questions makes it particularly useful for real software development, enabling users to develop with efficiency and efficacy.

ACKNOWLEDGMENTS

The authors would like to thank Sidharth Gulati for his outstanding collaboration in the productionization of the models, Rick Nayar for his invaluable support and guidance, and the anonymous reviewers for their insightful feedback.

COMPANY PORTRAIT

Please visit https://aws.amazon.com/about-aws to learn more about AWS.

PRESENTER BIO

Suthee is an applied scientist at Amazon Web Services, where he is part of the Code Science and Intelligence team responsible for building intelligent systems that improve developer experience. This includes a large-scale fine-tuning, multi-task representation learning, and in-house data labeling. Previously, he served as a core member of the data science team that developed a semantic search for a major e-commerce platform. He received his PhD in Computer Science from Santa Clara University where he specialized in deep learning techniques for search and recommendation.

REFERENCES

[1] Alon, U., Brody, S., Levy, O., and Yahav, E. (2019). code2seq: Generating sequences from structured representations of code. In *International Conference on Learning Representations*.

[2] Arabzadeh, N., Bigdeli, A., Seyedsalehi, S., Zihayat, M., and Bagheri, E. (2021). Matches made in heaven: Toolkit and large-scale datasets for supervised query reformulation. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, CIKM '21, page 4417–4425, New York, NY, USA. Association for Computing Machinery.

[3] Brandt, J., Guo, P. J., Lewenstein, J., Dontcheva, M., and Klemmer, S. R. (2009). Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, page 1589–1598, New York, NY, USA. Association for Computing Machinery.

[4] Chen, J., Mao, J., Liu, Y., Zhang, F., Zhang, M., and Ma, S. (2021). Towards a better understanding of query reformulation behavior in web search. In *Proceedings of the Web Conference 2021*, WWW '21, page 743–755, New York, NY, USA. Association for Computing Machinery.

[5] Guo, D., Lu, S., Duan, N., Wang, Y., Zhou, M., and Yin, J. (2022). UniXcoder: Unified Cross-Modal Pre-training for Code Representation. arXiv:2203.03850 [cs].

[6] Haiduc, S., Bavota, G., Marcus, A., Oliveto, R., De Lucia, A., and Menzies, T. (2013). Automatic query reformulations for text retrieval in software engineering. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 842–851, San Francisco, CA, USA. IEEE.

[7] Hill, E., Pollock, L., and Vijay-Shanker, K. (2011). Improving source code search with natural language phrasal representations of method signatures. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 524–527, Lawrence, KS, USA. IEEE.

[8] 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 *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5690–5700, Online. Association for Computational Linguistics.

[9] Iyer, S., Konstas, I., Cheung, A., and Zettlemoyer, L. (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, Berlin, Germany. Association for Computational Linguistics.

[10] Jain, P., Jain, A., Zhang, T., Abbeel, P., Gonzalez, J., and Stoica, I. (2021). Contrastive code representation learning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5954–5971, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

[11] Jinqiu Yang and Lin Tan (2012). Inferring semantically related words from software context. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 161–170, Zurich. IEEE.

[12] Keivanloo, I., Rilling, J., and Zou, Y. (2014). Spotting working code examples. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, page 664–675, New York, NY, USA. Association for Computing Machinery.

[13] Kim, B., Choi, H., Yu, H., and Ko, Y. (2021). Query reformulation for descriptive queries of jargon words using a knowledge graph based on a dictionary. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, CIKM '21, page 854–862, New York, NY, USA. Association for Computing Machinery.

[14] Lv, F., Zhang, H., Lou, J.-g., Wang, S., Zhang, D., and Zhao, J. (2015). CodeHow: Effective Code Search Based on API Understanding and Extended Boolean Model (E). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 260–270, Lincoln, NE, USA. IEEE.

[15] Nogueira, R., Lin, J., and Epistemic, A. (2019a). From doc2query to doctttttquery. *Online preprint*, 6.

[16] Nogueira, R., Yang, W., Lin, J., and Cho, K. (2019b). Document expansion by query prediction. *arXiv preprint arXiv:1904.08375*.

[17] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

[18] Sachdev, S., Li, H., Luan, S., Kim, S., Sen, K., and Chandra, S. (2018). Retrieval on source code: a neural code search. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 31–41, Philadelphia PA USA. ACM.

[19] Sridhara, G., Hill, E., Pollock, L., and Vijay-Shanker, K. (2008). Identifying Word Relations in Software: A Comparative Study of Semantic Similarity Tools. In *2008 16th IEEE International Conference on Program Comprehension*, pages 123–132, Amsterdam. IEEE.

[20] Stack Exchange, I. (2022). Stack exchange data dump.

[21] Xia, X., Bao, L., Lo, D., Xing, Z., Hassan, A. E., and Li, S. (2018). Measuring program comprehension: A large-scale field study with professionals. *IEEE Transactions on Software Engineering*, 44(10):951–976.

[22] Xiong, L., Xiong, C., Li, Y., Tang, K.-F., Liu, J., Bennett, P. N., Ahmed, J., and Overwijk, A. (2021). Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *International Conference on Learning Representations*.

[23] Zügner, D., Kirschstein, T., Catasta, M., Leskovec, J., and Günnemann, S. (2021). Language-agnostic representation learning of source code from structure and context. In *International Conference on Learning Representations (ICLR)*.