

Searching by Code: a New SearchBySnippet Dataset and SnippeR Retrieval Model for Searching by Code Snippets

Ivan Sedykh¹, Dmitry Abulkhanov², Nikita Sorokin¹,
Sergey Nikolenko^{3,4}, Valentin Malykh^{1,4}

¹ Huawei Noah's Ark lab, ² Independent Researcher,

³ St. Petersburg Department of the Steklov Institute of Mathematics,

⁴ Ivannikov Institute for System Programming
valentin.malykh@phystech.edu

Abstract

Code search is an important and well-studied task, but it usually means searching for code by a text query. We argue that using a code snippet (and possibly an error traceback) as a query while looking for bugfixing instructions and code samples is a natural use case not covered by prior art. Moreover, existing datasets use code comments rather than full-text descriptions as text, making them unsuitable for this use case. We present a new SearchBySnippet dataset implementing the search-by-code use case based on StackOverflow data; we show that on SearchBySnippet, existing architectures fall short of a simple BM25 baseline even after fine-tuning. We present a new single encoder model SnippeR that outperforms several strong baselines on SearchBySnippet with a result of 0.451 Recall@10; we propose the SearchBySnippet dataset and SnippeR as a new important benchmark for code search evaluation.

Keywords: code search, information retrieval, language model

1. Introduction

Increasing amounts of source code written every day lead to a plethora of possible issues, which almost inevitably have already been solved and reported upon on forums such as *StackOverflow*. A developer debugging an error has the relevant code snippet and error traceback produced by the compiler or interpreter, and she wants to find out the reasons behind the error and ways to fix it. This leads to the setting that we call “search by snippet”: based on a code snippet and/or error traceback, find posts that might contain a solution. To our surprise, this setting has been very scarcely considered in literature; e.g., [Ponzanelli et al. \(2014\)](#) consider it informally and just use a commercial search engine. In this work, we propose an information retrieval setup where the query is a code snippet and/or traceback and documents are posts with text and possibly other code snippets (Fig. 1); this setting can be automated and incorporated into IDEs. Previous works on code search (see Section 2) usually matched the source code of a function and its comment, and code search also considers text queries; one can invert the problem but the text parts are usually short comments rather than full-text posts that could contain a solution.

In this work, we present a new dataset called SearchBySnippet that captures this problem setting based on *StackOverflow* posts (in *Python*). We have adapted several state of the art code search models as baselines, including CodeBERT, GraphCodeBERT (GCB), and SynCoBERT. To our surprise, their performances on SearchBySnippet are

very poor; even GCB specially trained on the CodeSearchNet dataset for this setting lost very significantly to the simple BM25 baseline. Therefore, we have developed a new SnippeR model that uses a GCB-based encoder for both queries and documents and incorporates a number of improvements so it outperforms BM25 on SearchBySnippet. Still, absolute values of the results are not too high, and we believe that the problem setting embodied in SearchBySnippet opens up a new research direction that could lead to better code understanding.

The primary contributions of this work include: (i) a novel problem setting for code search and a new SearchBySnippet dataset for training and evaluation in this setting; (ii) the SnippeR model that outperforms strong information retrieval baselines and can serve as a starting point for research in this new setting¹. Below, Section 2 surveys related work, Section 3 presents SearchBySnippet, Section 4 introduces SnippeR and its training procedure, Section 5 shows our experimental setup and results, and Section 6 concludes the paper.

2. Related Work

Datasets. [Husain et al. \(2019\)](#) presented *CodeSearchNet* (CSN), constructed from a *GitHub* dump, with function bodies split into the code itself and a description. CSN contains 2M (code snippet, description) pairs in 6 programming languages, including *Python*. [Hasan et al. \(2021\)](#) combined CSN

¹We are going to release the SearchBySnippet and SnippeR source code once the clearance is done.

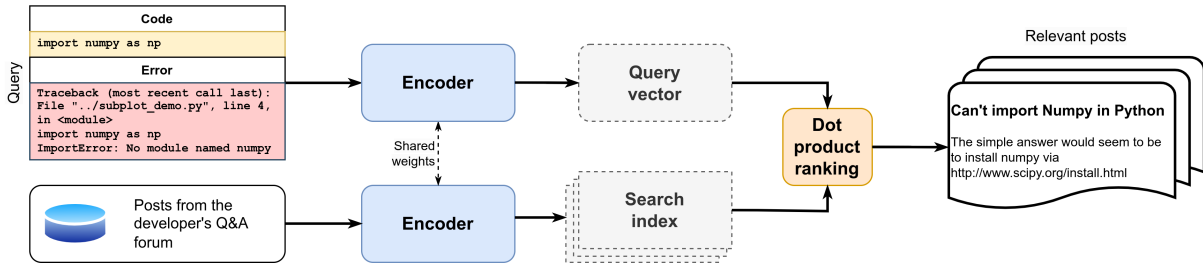


Figure 1: Overview of the problem setting and system design.

and other datasets into a larger one (with *Java* and *Python* subsets of CSN), getting 4M (code snippet, description) pairs. An even larger dataset had been constructed earlier by Gu et al. (2018); their *CODEnn-Train Java*-based dataset has 18M pairs of methods and their one-sentence descriptions. *CodeXGLUE* by Lu et al. (2021) is a machine learning benchmark collection of datasets for code understanding and generation tasks, which includes code in 10 programming languages (and a modification of CSN). Another multi-task dataset was presented by Puri et al. (2021), with 14M code snippets in 5 programming languages.

Code Search. Dense vector representations are often used for information retrieval (IR): Gu et al. (2018) used two RNNs to represent the code and textual descriptions, Feng et al. (2020) based CodeBERT on language models, Gotmare et al. (2021) used three Transformer-based models, two encoders and one classifier, to obtain a hierarchical representation of code and text. Our model uses a single encoder for embedding both queries and documents and has no separate classifier.

Language models for code. GraphCodeBERT (Guo et al., 2021) uses data flow graphs during pretraining to solve masked language modeling, edge prediction, and node alignment tasks. SynCoBERT (Wang et al., 2021) uses multimodal contrastive learning to achieve better code representations and is pretrained on identifier prediction and abstract syntax tree (AST) edge prediction.

3. SearchBySnippet Dataset

Data Preprocessing. SearchBySnippet is constructed from a public *StackOverflow* dump² with questions and answers from 2014 to 2021 and rich meta-information. During submission, *StackOverflow* users fill in several fields that appear in the dump structure (along with fields such as “FavouriteCount”); the “tags” field allows to easily categorize questions. We limit our work to *Python* due to its popularity. For preprocessing, we take the “text” and “title” fields that contain the main text of a question (“text” can have formatting

²<https://archive.org/details/stackexchange>

Constraint	Number	%
SearchBySnippet, total	948 749	100.00
with “code”	670 561	70.68
with “error”	478 992	50.49
with “code” and “error”	342 086	36.06
with “code” or “error”	807 467	85.10
with “best_answer”	337 797	35.60
CodeSearchNet, total	2 070 536	200.18
Python only	457 461	48.22
NeuralCodeSearch, evaluation	287	< 0.01

Table 1: SearchBySnippet dataset statistics and comparison with other code search datasets.

markup) and the `<code>` tag for source code and/or system output and extract text from these tags. If it does not look like a traceback (e.g., does not have the “Error” keyword), we mark it as “code” and extract in the “code” field; if it does, we use the “error” field. We also parse the error type from the traceback with regular expressions and put it into the “keyword” field. If a question contains several `<code>` tags, they are classified independently. We add a “best_answer” field for the answer accepted by the user and store the original text in the “body” tag. Fig. 1 shows a sample preprocessed query on the left and a document on the right.

Table 1 shows the dataset statistics. Only half of the questions have error tracebacks, and over 70% contain code. Interestingly, questions where we have extracted both “code” and “error” fields cover only 1/3 of the dataset, while the ones with “code” or “error” cover 85%. Only 35% of the questions have accepted answers. Table 2 shows the average sizes (in symbols) for extracted fields and compares them with the “body” field (percentages only count questions where the field(s) are present).

Evaluation Set. Some questions in the *StackOverflow* dump are marked as duplicates; usually post A is a duplicate of post B if *StackOverflow* moderators have deemed the question in post A to be equivalent to the question in post B. We selected duplicated questions that contain an accepted answer (in post B) and a code snippet or traceback, getting 1369 questions that we use for

Subset	Total size	Field size	% of total
SearchBySnippet			
All posts	1340.18	1340.18	100.00
with "code"	1630.62	576.26	35.34
with "error"	1981.50	681.04	34.37
with "code" and "error"	2223.89	1366.80	61.46
with "best_answer"	1297.66	880.98	67.89
CodeSearchNet (% of SearchBySnippet)			
"func_code_string"		755.59	55.38
Python only		1060.22	77.57
"func_documentation_string"		209.87	23.82
Python only		297.59	33.78
NeuralCodeSearch (% of SearchBySnippet)			
Answer		159.75	11.59
Question		50.69	5.75

Table 2: Average field sizes, in symbols; "Field size" shows the size of the field in "Subset".

evaluation. We use a union of the "code" and "error" fields from post A as query and "best_answer" from post B as the ground truth document.

Comparison to Other Datasets. We compare our dataset to CodeSearchNet (Husain et al., 2019) also devoted to code search. It contains snippets in several programming languages, including *Python*, in the form of functions paired with their descriptions. We also consider *NeuralCodeSearch* (Li et al., 2019) as the dataset with the most similar design; we only use its evaluation part of this dataset that contains *StackOverflow* questions with code snippets cut out from the best answers.

Tables 1 and 2 compare *CodeSearchNet* (CSN) and *NeuralCodeSearch* (NCS) with *SearchBySnippet*. CSN is twice larger overall, but its *Python* part is twice as small as *SearchBySnippet*. NCS contains only 287 questions in its evaluation part, 3500x less than *SearchBySnippet*. Table 2 compares CSN and NCS with *SearchBySnippet* in terms of the average size of various fields (in symbols); we assume that "func_code_string" in CSN is a rough equivalent of the union of our "code" and "error", and "func_documentation_string" corresponds to "best_answer". For NCS, the "answer" and "question" fields are inverted since "best_answer" in our case is a text field, while in NCS it is a code snippet. CSN and NCS parts of Table 2 show percentages of the corresponding (code and text) average field sizes in *SearchBySnippet*; while code-containing fields in CSN are only 20% shorter, the text field is 3x to 4x times shorter than in *SearchBySnippet*. We believe that this could make retrieval on *SearchBySnippet* more difficult. In NCS, both entities are an order of magnitude shorter, leading to a much easier retrieval task.

4. Model

Problem setting. In our IR task, the query is a code snippet and/or traceback from the "code" and "error" fields and documents are answers from the "best_answer" field. Given a collection of documents D and a query q , the model has to rank documents so that the ground truth answer $d \in D$ is closer to the beginning of the list. Following prior art on code search, we use a neural network encoder to obtain dense vector representations of queries and documents. Unlike Karpukhin et al. (2020), and following Feng et al. (2020) in code-related tasks and Sorokin et al. (2022) in general IR, we use the same encoder E for both queries and documents. The system first encodes all documents in the database into embedding vectors and then constructs the search index. For a query q , it computes pairwise similarity scores between $E(q)$ and document embeddings $E(d)$ and sorts them with the dot product score $\text{score}(q, d) = E(q)^\top E(d)$. Fig. 1 shows the system structure; we call this model **SnippeR** (Snippet Retrieval).

We initialize the encoder E with pretrained *GraphCodeBERT* (GCB) (Guo et al., 2021), a model based on RoBERTa (Liu et al., 2019) with 125M trainable parameters, pretrained for source code using a data flow graph along with the text representation. In our case the input is not always code, so we cannot use the data flow graph, but we discovered that even without it GCB outperforms other models. We used the model output (last layer's hidden state) for the first $\langle s \rangle$ token as a vector representation of the input text (or code).

Training procedure. The encoder is trained to maximize the similarity between a query and the matching document's embedding while minimizing the similarity between a query and embeddings of irrelevant documents. Each training sample contains one query q , one relevant (positive) document d^+ , and n irrelevant (negative) documents $D^- = \{d_j^-\}_{j=1}^n$. As the contrastive loss we use the negative log-likelihood of the positive document:

$$\mathcal{L}(q, d^+, D^-) = -\log \frac{e^{\text{score}(q, d^+)}}{\sum_{j=1}^n e^{\text{score}(q, d_j^-)} + e^{\text{score}(q, d^+)}}$$

For training, we use *hard negatives* mined from the previous model iteration via *self-training*. We use iterative learning (Qu et al., 2021; Izacard and Grave, 2020) in the form shown in Fig. 2: on each step, the model retrieves top k documents from the database for every training set query. Then we treat these top k documents (except for the ground truth answer) as hard negative examples for the next model training iteration; in Section 5 we show the results after one such loop (SnippeR₂). For each query, as other (non-hard) negatives we use other documents from the training batch and their hard negative samples; this is the *in-batch negative*

trick (Karpukhin et al., 2020; Sorokin et al., 2022) that helps avoid sampling additional negatives.

Pretraining. SearchBySnippet has only 1369 questions with duplicates and accepted answers in the evaluation part, but 148K pairs of duplicate questions with code snippets or tracebacks but no accepted answers. We use them in pretraining to better adapt the model for the structure and semantics of *StackOverflow*. Pretraining runs the same as training with two differences: (i) for a pair of duplicates A and B we use the snippet and/or traceback from A as the query and B as the target document; (ii) we include post bodies in the texts since they do not overlap with the evaluation set.

Data preprocessing and training setup. We concatenate the code snippet c and traceback t (“code” and “error” fields) to form a query $q = [c, t]$. Queries are often longer than maximum input length (256 or 512 tokens), and since the end of a traceback usually contains crucial information such as error identifiers and meaningful error descriptions, we remove tokens from the middle rather than the end, leaving equal number of tokens at the beginning and end. Text documents are truncated to (the first) 256 or 512 tokens. In SearchBySnippet, documents are represented as question title, question body, and accepted answer (“title”, “body”, and “best_answer” fields). Since queries were extracted from post bodies, we cannot use them in the “body” field in the training set, so the post body was removed from a document representation during training, leaving the model with “title” and “best_answer” fields for a document. In evaluation, we use the “body” field as well since now there is no issue with leaking the answer.

5. Evaluation

Setup and hyperparameters. We measure model performance with $\text{Recall}@k = \sum_{i=1}^k [r_i = d^+]$, where d^+ is the ground truth document and r_i is the document retrieved at position i ; $k \in \{5, 10, 20, 50\}$ in our experiments. The model was trained for 21 hours on 2 NVIDIA Tesla V100 GPUs (16GB memory each). We used the Adam optimizer (Kingma and Ba, 2014) with constant learning rate schedule and 3500 warm-up steps. To stabilize training we clipped the gradient norm to 2.0. The learning rate was set to 10^{-5} , batch size 12.

Baselines. We use the standard information retrieval baseline *Okapi BM25* (Robertson et al., 1994). The dataset has a significant distribution shift between training and evaluation; since BM25 does not train, it does not suffer from the shift, which is an important factor making this baseline strong. Other baselines include modern Transformer-based pretrained models for NLP and code understanding, trained to produce meaningful

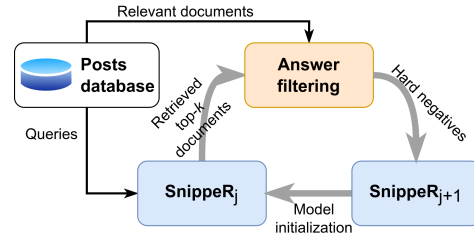


Figure 2: Self-training framework.

Model	Recall			
	@5	@10	@20	@50
GraphCodeBERT (Guo et al., 2021)	0.001	0.001	0.002	0.009
CodeBERT (Feng et al., 2020)	0.001	0.006	0.010	0.013
SynCoBERT (Wang et al., 2021)	0.006	0.010	0.013	0.020
GraphCodeBERT (+CSN)	0.161	0.221	0.280	0.367
BM25 (Robertson et al., 1994)	0.311	0.406	0.474	0.562
SnippeR	0.338	0.451	0.536	0.657

Table 3: Results on SearchBySnippet.

vector representations for code and/or text in the context of code search (Husain et al., 2019); e.g., CodeBERT (Feng et al., 2020) aims to align the embeddings of the code and corresponding text. We also evaluated GraphCodeBERT (GCB) (Guo et al., 2021) and SynCoBERT (Wang et al., 2021) that incorporate abstract syntactic tree (AST) representations for code. ASTs are used in training but not inference since short snippets from queries often do not yield a meaningful AST. We considered these models as base models for SnippeR in preliminary experiments, and GCB won. We also tried to fine-tune GraphCodeBERT on CodeSearchNet; since our data is noisy, we fine-tuned GraphCodeBERT without ASTs (“GraphCodeBERT (+CSN)”). All models in Table 3 (except BM25) are based on RoBERTa, with 125M trainable parameters.

Results. Our main evaluation results are shown in Table 3. Surprisingly, all Transformer-based models perform very poorly out of the box and lose to the classic BM25 baseline, despite the fact that they have been trained to embed both source code and text into a single embedding space. Fine-tuning GCB on CSN significantly improved performance, but even then GCB falls short of BM25 by a large margin. We present the best result for SnippeR in the table; it has been able to outperform BM25 and all other baselines by all considered metrics. Still, the resulting Recall@5 only slightly exceeds 30% and Recall@50 is about 65%, which leaves significant room for improvement.

6. Conclusion

We have presented a novel use case for code search that has not been widely studied in literature: searching *by* a code snippet and/or error

traceback. We have presented a novel way to construct a dataset for this use case, leading to the SearchBySnippet dataset with about 1M queries. We have evaluated several code understanding models and found that on SearchBySnippet they all lose even to the BM25 baseline. Thus, we have developed a new model SnippeR for searching by code snippets and tracebacks, and have been able to outperform BM25 on SearchBySnippet. Still, absolute values of our results are relatively low, and we hope that this new setting and dataset will serve as a new research direction for code understanding, with SnippeR providing a reasonable baseline.

Acknowledgements

The authors are grateful to colleagues from Huawei Noah's Ark lab, especially to Irina Piontkovskaya and Wang Yasheng for organization of the collaboration which allowed this paper to happen. The work of Sergey Nikolenko and Valentin Malykh was supported by a grant for research centers in the field of artificial intelligence, provided by the Analytical Center for the Government of the Russian Federation in accordance with the subsidy agreement (agreement identifier 000000D730321P5Q0002) and the agreement with the Ivannikov Institute for System Programming of the Russian Academy of Sciences dated November 2, 2021, No. 70-2021-00142.

7. Bibliographical References

- Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. [Codebert: A pre-trained model for programming and natural languages](#).
- Akhilesh Deepak Gotmare, Junnan Li, Shafiq Joty, and Steven CH Hoi. 2021. Cascaded fast and slow models for efficient semantic code search. *arXiv preprint arXiv:2110.07811*.
- 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, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. [GraphCodeBERT: Pre-training Code Representations with Data Flow](#). Technical Report arXiv:2009.08366, arXiv. ArXiv:2009.08366 [cs] type: article.
- Masum Hasan, Tanveer Muttaqueen, Abdullah Al Ishtiaq, Kazi Sajeed Mehrab, Md Mahim Anjum Haque, Tahmid Hasan, Wasi Ahmad, Anindya Iqbal, and Rifat Shahriyar. 2021. Codesc: A large code–description parallel dataset. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 210–218.
- Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. [CodeSearchNet Challenge: Evaluating the State of Semantic Code Search](#).
- Gautier Izacard and Edouard Grave. 2020. [Distilling Knowledge from Reader to Retriever for Question Answering](#). Number: arXiv:2012.04584 arXiv:2012.04584 [cs].
- Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. [Dense passage retrieval for open-domain question answering](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.
- Diederik P. Kingma and Jimmy Ba. 2014. [Adam: A method for stochastic optimization](#). Cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.
- Hongyu Li, Seohyun Kim, and Satish Chandra. 2019. Neural code search evaluation dataset. *arXiv preprint arXiv:1908.09804*.
- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. [Roberta: A robustly optimized bert pre-training approach](#).
- Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.
- Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. 2014. Mining stackoverflow to turn the ide into a self-confident programming prompter. In *Proceedings of the 11th working conference on mining software repositories*, pages 102–111.

- Ruchir Puri, David S Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, et al. 2021. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.
- Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. 2021. [RocketQA: An Optimized Training Approach to Dense Passage Retrieval for Open-Domain Question Answering](#). In *NAACL*.
- Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gattford. 1994. Okapi at trec-3. In *TREC-3*, pages 109–126.
- Nikita Sorokin, Dmitry Abulkhanov, Irina Piontkovskaya, and Valentin Malykh. 2022. Ask me anything in your native language. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 395–406.
- Xin Wang, Yasheng Wang, Fei Mi, Pingyi Zhou, Yao Wan, Xiao Liu, Li Li, Hao Wu, Jin Liu, and Xin Jiang. 2021. [SynCoBERT: Syntax-Guided Multi-Modal Contrastive Pre-Training for Code Representation](#). Number: arXiv:2108.04556 arXiv:2108.04556 [cs].