Selma: A Semantic Local Code Search Platform

Anja Reusch, Guilherme C. Lopes, Wilhelm Pertsch, Hannes Ueck, Julius Gonsior, Wolfgang Lehner

Database Systems Group, Technische Universität Dresden, Germany
`{firstname.lastname}@tu-dresden.de`

Abstract. Searching for the right code snippet is cumbersome and not a trivial task. Online platforms such as Github.com or searchcode.com provide tools to search, but they are limited to publicly available and internet-hosted code. However, during the development of research prototypes or confidential tools, it is preferable to store source code locally. Consequently, the use of external code search tools becomes impractical. Here, we present Selma[1]: a local code search platform that enables term-based and semantic retrieval of source code. Selma searches code and comments, annotates undocumented code to enable term-based search in natural language, and trains neural models for code retrieval.

Keywords: Code Retrieval · Transformer Models.

1 Introduction

Software development plays a key role in many aspects of modern life. In this context, platforms like StackOverflow.com play a huge role in the daily work of software developers and researchers. According to a recent survey[2], at least 80% of them visit Stack Overflow a few times a week, 50% even daily. This fact points to the importance of code search for software development. Especially, when new developers join existing projects, they undergo a steep learning curve, since internal code is often not as easily searchable as well-known libraries. In order to enable the search in these internal projects, online tools can usually not be applied since the project and its code need to be kept confidential.
Therefore, we present Selma, a semantic code search platform that runs within a local environment. Semantic code search refers to the task of using natural language to formulate a query in order to search in documents written in source code. At its core, it consists of two code retrieval models, one traditional BM25 method and a Transformer-Encoder [14]. Transformer-Encoder models, of which BERT [2] is the most prominent approach, have already demonstrated great success in modeling natural languages and were therefore incorporated into current retrieval systems (for a survey, see [15]). Apart from natural language, Transformer-Encoders were also trained for source code understanding [7, 4, 5,

[1] Code and Videos: https://anreu.github.io/selma
[2] https://survey.stackoverflow.co/2022/

11, 9]. Also, models using Transformer-Decoders for code were developed, for which GitHub Co-Pilot[3] is one example. These models are trained to generate code given a prompt in natural language. We, however, want to rely on code that is actually existent in the code base, and not on code that is only based on some code snippets the model was trained on. In addition, a recent study by Nguyen et al. [12] found that the generated code is written correctly in only half of the evaluated cases. Therefore, we decided to combine CodeBERT [4], a Transformer-Encoder trained for source code understanding instead of generation, with ColBERT [8] to enable fast retrieval times.
Selma also includes code expansion, where documentation is generated for undocumented code snippets to enable natural language queries also for BM25. Different configurations of retrieval mechanisms and indexes can be applied depending on the code base and hardware setup. A preview of the platform can be seen in Fig. 1. In the following, we will present a walk-through of the system from the perspective of a user and provide an overview of the different components Selma is built of.

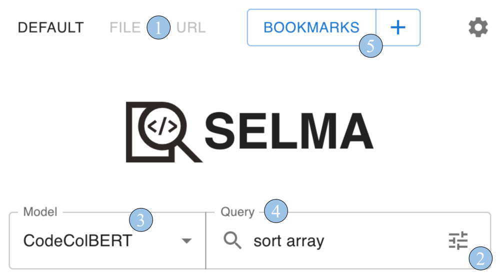
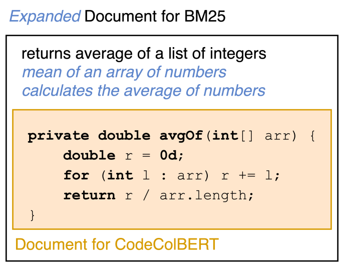


Fig. 1: Screenshot of Selma.


Expanded Document for BM25

returns average of a list of integers
mean of an array of numbers
calculates the average of numbers

```
private double avgOf(int[] arr) {
    double r = 0d;
    for (int l : arr) r += l;
    return r / arr.length;
}
```

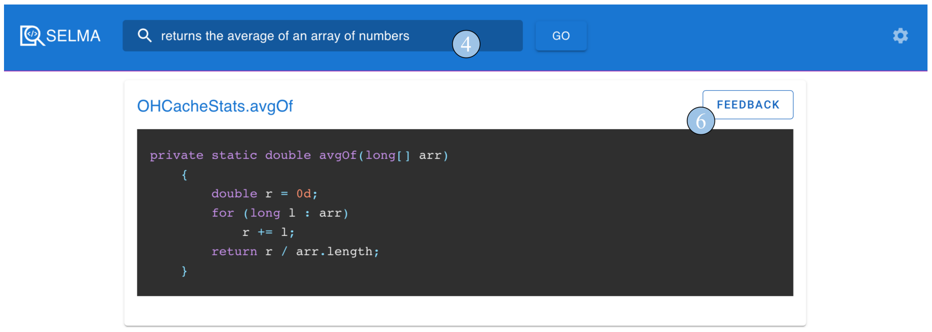
Document for CodeColBERT

Fig. 2: Code expansion.

2 System Overview

In this section, we present the main features of Selma. Fig. 1 and 3 show screenshots of our platform.
The system first needs to be set up for a new code base. During the set up process, a Git repository URL is entered, which will then be cloned automatically (1). The user who sets up the system - usually an administrator - can decide which of the two retrieval methods are used to build the index: the term-based BM25 and the Transformer-Encoder-based CodeColBERT. The BM25 index is built using the code snippet and its documentation as an index-able document, while CodeColBERT is applied directly on the code snippets. Due to CodeColBERT's high code understanding capabilities, this index delivers the best results.

[3] https://github.com/features/copilot



```
private static double avgOf(long[] arr)
    {
        double r = 0d;
        for (long l : arr)
            r += l;
        return r / arr.length;
    }
```

Fig. 3: Result preview for the query "returns the average of an array of numbers".

It requires, however, a GPU to run, which is why we also offer the BM25 index as a second option. To provide semantic search even when using the BM25 index, the administrator can choose to use SELMA's code expansion feature, which runs two Transformer models that generate documentation. The system benefits from the generated documentation since it expands the document by additional terms that were originally not included in the documentation (see Fig. 2). Depending on the chosen set up, in the background the BM25 index is constructed or the CodeColBERT model is either trained or applied directly to compute and store the index for the new repository. When adding a new repository, the administrator can choose to add it to an existing index or to create the index from scratch. When choosing an existing index, the settings of this index are applied to index the new code snippets. After the index is built, the administrator selects an index (2) and a model (3). Now, the user of the system can enter a query in natural language (4). They can also choose a query from the query bookmark menu (5), which stores frequently executed queries. A screenshot of the results page can be found in Fig. 3. For each result returned by the selected model, the user can provide feedback on the relevance of the code snippet (6).

Pre-Processing After cloning a code repository, we use the tool Treesitter[4] to parse the code, split it into method level granularity and extract existing documentation. Tokenization is done by each method separately: For term-based retrieval the internal tokenizer of PyTerrier is used. Each transformer-based model uses its respective tokenizer.

Retrieval Methods As a base retrieval method, we employ BM25 as it is offered as part of PyTerrier [10] with $k_1 = 1.2$, $k_3 = 8$, and $b = 0.75$. The code snippets are concatenated with the documentation string to serve as documents. We found that using the method name, the parameter names, and the returned value as a representation for the code snippet works best. We chose ColBERT [8] as our second retrieval method, because it combines the semantic modeling capacities of Transformer-Encoders with fast retrieval performance due to offline

[4] https://tree-sitter.github.io/tree-sitter/

indexing. This is achieved by passing each document through the Transformer-Encoder and storing its embeddings. At query processing time, the query is also passed through the encoder to compute the query embeddings, which are then used to determine the relevance between query and each document embedding. The documents with the highest relevance scores are returned by the system. As a base model for ColBERT we employ CodeBERT that models English and six programming languages [4]. We fine-tune the model for code retrieval using the CodeSearchNet Data Set [6], where the documentation comment, which describes a function, serves as the query and the method body of a code snippet serves as the document. Negative examples are sampled randomly from the entire corpus. Our CodeColBERT model can be used directly to index new documents, but it can also be adapted by further training on examples from new repositories. In addition, if the user provides positive and negative feedback for retrieval results, these can also serve as training examples. For retrieval, we use the PyTerrier bindings for ColBERT.

Code Expansion Since not all code is always documented perfectly, SELMA can automatically add documentation for undocumented code snippets. The idea is inspired by Nogueria et al.'s Doc2Query [13] where a query is predicted for a given document. The predicted query is then appended to the documents which is then indexed. Our idea is similar: We generate documentation strings that will be included when indexing code snippets. This way we bridge the language mismatch between query and documents. For documentation generation, we include two models in the system: CodeTrans [3] and PLBART [1], which are both based on the Transformer architecture [14] and are trained on several tasks dealing with source code. We use the models fine-tuned on code summarization translating from source code to English. We provide the method body to the models, which then produce a documentation string. This string is concatenated with the method body to serve as the expanded document in the index. Experiments on the CodeSearchNet Challenge verified that BM25 with Code Expansion is almost on par with CodeColBERT while executing queries ten times faster.

3 Conclusion

This work presented SELMA, a semantic code search platform designed for local use. Its unique features are: term-based and neural code search, Code Expansion facilitated by Transformer models, and an integrated user feedback for both evaluation and refinement. We provide an already fine-tuned model[5] ready for immediate application in code search tasks without the need for further training, but it can also be adapted to the code that needs to be searched. This work was supported through grant 01IS17044 Software Campus 2.0 (TU Dresden) by the BMBF. Furthermore, the authors are grateful for the GWK support for funding this project by providing computing time through the Center for Information Services and HPC (ZIH) at TU Dresden.

[5] https://huggingface.co/ddrg/codecolbert

References

1. Ahmad, W.U., Chakraborty, S., Ray, B., Chang, K.W.: Unified pre-training for program understanding and generation. arXiv preprint arXiv:2103.06333 (2021)
2. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. 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). pp. 4171–4186. Association for Computational Linguistics, Minneapolis, Minnesota (Jun 2019). https://doi.org/10.18653/v1/N19-1423, https://aclanthology.org/N19-1423
3. Elnaggar, A., Ding, W., Jones, L., Gibbs, T., Feher, T., Angerer, C., Severini, S., Matthes, F., Rost, B.: Codetrans: Towards cracking the language of silicon's code through self-supervised deep learning and high performance computing. arXiv e-prints pp. arXiv–2104 (2021)
4. Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., et al.: Codebert: A pre-trained model for programming and natural languages. arXiv preprint arXiv:2002.08155 (2020)
5. Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., Fu, S., et al.: Graphcodebert: Pre-training code representations with data flow. In: ICLR (2021)
6. Husain, H., Wu, H.H., Gazit, T., Allamanis, M., Brockschmidt, M.: Codesearchnet challenge: Evaluating the state of semantic code search. arXiv preprint arXiv:1909.09436 (2019)
7. Kanade, A., Maniatis, P., Balakrishnan, G., Shi, K.: Learning and evaluating contextual embedding of source code. In: International Conference on Machine Learning. pp. 5110–5121. PMLR (2020)
8. Khattab, O., Zaharia, M.: Colbert: Efficient and effective passage search via contextualized late interaction over bert. In: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 39–48 (2020)
9. Liu, S., Wu, B., Xie, X., Meng, G., Liu, Y.: Contrabert: Enhancing code pre-trained models via contrastive learning. arXiv preprint arXiv:2301.09072 (2023)
10. Macdonald, C., Tonellotto, N.: Declarative experimentation ininformation retrieval using pyterrier. In: Proceedings of ICTIR 2020 (2020)
11. Neelakantan, A., Xu, T., Puri, R., Radford, A., Han, J.M., Tworek, J., Yuan, Q., Tezak, N., Kim, J.W., Hallacy, C., et al.: Text and code embeddings by contrastive pre-training. arXiv preprint arXiv:2201.10005 (2022)
12. Nguyen, N., Nadi, S.: An empirical evaluation of github copilot's code suggestions. In: Proceedings of the 19th International Conference on Mining Software Repositories. p. 1–5. MSR '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3524842.3528470, https://doi.org/10.1145/3524842.3528470
13. Nogueira, R., Yang, W., Lin, J., Cho, K.: Document expansion by query prediction. arXiv preprint arXiv:1904.08375 (2019)
14. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
15. Yates, A., Nogueira, R., Lin, J.: Pretrained transformers for text ranking: Bert and beyond. In: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 2666–2668 (2021)