Humboldt-Universität zu Berlin
Institut für Informatik

Retrieving code for question answering on scientific workflows

Master Thesis Expose

Simon Bosse

September 30, 2025

1 Introduction

In many scientific areas, the volume of available data is steadily increasing. To ensure large amounts of scientific data are processed within reasonable amounts of time, data analysis is often executed on distributed systems. Also, analyzing data from domains such as genomics or earth observation often requires multiple interconnected analysis steps. The use of workflow systems can both improve reusability and help to exploit the computing power of distributed systems. First, such workflows are implemented in domain-specific languages, which facilitate the specification of a step-by-step analysis. Afterwards, scientific workflow management systems such as Nextflow [6] or Snakemake [16] enable workflow execution in different environments, offering integration with systems like Kubernetes[1] or Slurm [35].

Developing scientific workflows is a challenging task located at the intersection of multiple domains: While the selection and chaining of analysis steps requires domain-specific knowledge, software engineering skills are needed to subsequently implement the developed analysis chain as a scientific workflow. For repeatedly occurring scientific analysis goals, there often exist best-practice workflows, e.g. those developed and maintained by the nf-core [7] community. However, the deployment of these best-practice workflows is by no means trivial: nf-core code adheres to many standards and integrates a multitude of different analysis options. This increases the complexity of the code, which can overwhelm domain scientists without prior software engineering knowledge. Also, coding errors or lack of maintenance are common reasons for workflows to crash [27].

To be able to deploy or adapt existing scientific workflows, a good understanding of the code is crucial. In recent years, LLM-based methods have emerged as a new tool to facilitate various software development tasks such as inline code completion or code reviews [11]. To accelerate the comprehension of large code bases, code-based question answering (QA) systems can be deployed, answering questions on given programming code. Although some popular LLMs such as GitHub Copilot[2] also integrate a chat function enabling users to ask code-related questions, language models have been found to struggle with questions in broader code contexts, e.g. on repository level [3]. Especially for scientific workflow languages, reliable code

[1] https://kubernetes.io/
[2] https://github.com/features/copilot

QA systems could facilitate the deployment of best-practice workflows by domain scientists.

In prior work, we developed a QA dataset on 10 open-source Nextflow workflows [1]. The evaluation of SOTA language models confirmed the previously mentioned limitations of LLMs for questions on longer code contexts: Model performance significantly declined for those workflows with the most lines of code (LOC). Inspired by current RAG systems, this thesis aims to address such issues by implementing a retriever, which selects relevant code snippets for a given question. The retrieved passages can then be supplied to the LLM instead of the complete workflow, reducing the context length and sparing the LLM the task of retrieving the relevant code passages for a given question. Also, we aim to extend the existing QA dataset on Nextflow code.

2 Related Work

2.1 Retrieval

Information Retrieval (IR) can be described as the task of retrieving relevant information (for a given user query) from a reference corpus. Aiming to address such tasks, retrieval systems are integrated within various applications like search engines and modern language models [10]. Retrieval-Augmented Generation (RAG) provides large language models with additional information that is retrieved beforehand for a given prompt [18]. In coding-related applications, retrieval systems often need to process and connect both programming code and natural language [13]. Subsequently, several different types of code retrieval can be defined: For instance, text-to-code retrieval includes problems such as retrieving the SQL query matching a given natural language question [20].

Two popular approaches to address IR tasks are sparse and dense retrieval. Sparse retrievers measure word overlap between query and reference documents using bag-of-words (BOW) representations [25]. These representations of queries and documents consist of high-dimensional, sparse vectors. There exist several strategies to weight BOW representations: For example, Term Frequency-Inverse Document Frequency (TF-IDF) techniques decrease the weights of words that occur more commonly across the reference documents [28].

A core problem of sparse retrievers is their reliance on syntactical equality between matching words, failing to capture semantical resemblances [15]. This limitation can be addressed using dense retrieval: Here, queries and documents are encoded to dense low-dimensional vectors using pre-trained language models [39]. For retrieval, the similarity between these embeddings needs to be scored [38], e.g. by using the inner product [25]. Performance-wise, hybrid systems incorporating both sparse and dense retrieval have been found to be most effective [25, 30].

There exist various techniques to further improve the performance of RAG systems: Query rewriting alters queries before retrieval, aiming to generate a query more targeted on the information needed [26]. Also, supplying long contexts to LLMs can be costly since inference cost increases with input length. To reduce this effect, documents can be compressed: Extractive compression solely selects relevant sentences, effectively pruning irrelevant information. Abstractive compression aims to summarize and reformulate documents with respect to the query [32].

Compared to natural language, code is constrained to a fixed set of syntactic constructs. Code can therefore be treated as structured information: The syntactic structure itself contains semantical information [22]. This structure can also be used to enhance retrievers in coding contexts. For code completion on repository level, researchers have used static code analysis methods such as dataflow analysis to improve the retrieval of relevant code context [5]. Another approach built graphs from textual queries as well as from code snippets, and then performed semantic graph matching to retrieve code matching a natural language description [22].

Other challenges can arise when information for multi-hop questions needs to be retrieved. Such questions require multi-hop reasoning, i.e. the connection of information across multiple documents [34]. Since some of the documents may depend on information from other documents rather than the question itself, not all relevant information can certainly be retrieved using only the question itself.

An approach to deal with such challenges is retrieving relevant documents not in one step, but in an iterative manner. Researchers have trained a system retrieving items iteratively, selecting new documents in each step while keeping the previously selected items in mind [4]. Compared to standard retrieval, this approach also considers the relation between the selected documents. In another study, the idea of an iterative retriever was combined with the principles of beam search: After each step, the k most promising

retrieved sets of documents were kept as candidates for the next step, instead of just retrieving one document in each step [37].

Other researchers used an iterative retrieval approach for code completion: Preceding code is not always directly relevant to the code that needs to be written. Therefore, using this preceding code as a retrieval query may not yield the anticipated results. The first attempt of code completion subsequently can be incorrect, e.g. when APIs or functions that had not been retrieved are called in a wrong way. In such cases, these incorrect attempts of code completion were expected to serve as a better retrieval query and therefore used in the next retrieval step [36].

In the area of scientific workflows, retrieving similar workflows can be of interest to increase reusability or facilitate workflow rewriting. Therefore, various publications also applied retrieval techniques to assess similarity between scientific workflows. Different structure-based measures, such as mapping modules or pairwise module comparison, as well as annotation-based measures like BOW with workflow descriptions, were compared in a study regarding their performance of retrieving similar scientific workflows.

When evaluating similarity using information on the module level, approaches using edit-distances of module labels outperformed those using exact matching. Also, techniques which estimate workflow similarity utilizing structural information proved able to outperform annotation-based measures in certain configurations [29]. More recently, researchers combined text embeddings with workflow embeddings (derived from the workflows structure), and afterwards used a deep learning model to obtain a similarity score of a scientific workflow for a given user query [8].

2.2 QA benchmarks for coding

There exist several datasets consisting of questions on programming code. CodeQA [23] consists of questions on python and Java code. The authors constructed QA pairs from in-line documentation, using rule-based and template-based methods. CS1QA [17] contains QA pairs which were gathered in an introductory python class.

The questions in ProCQA [21] were scraped from StackOverflow. Here, the authors defined a second task next to classic QA: In an answer retrieval task, a retriever was asked to find the correct answer for a given question from a set of answers. InfiBench [19] was also created by manually filtering high-quality questions from StackOverflow. CoReQA [3] and CodeRepoQA

[12] incorporate QA pairs on repository level instead of questions on small code snippets.

In the area of code retrieval, RepoBench [24] dissects code completion on repository level in a retrieval and a subsequent generation task. Furthermore, the CoIR dataset [20] integrates multiple code retrieval tasks on various datasets. Next to QA on code, three further types of code retrieval are defined:

1. **Text-to-Code Retrieval** consists of task such as retrieving code snippets for given web queries, or matching SQL queries to a given question in text.

2. The task of retrieving summaries for a given code snippet is described as **Code-to-Text Retrieval**.

3. Last, the **Code-to-Code Retrieval** task includes problems such as retrieving adjacent code for a given snippet, or finding the correct translation of a program in another coding language.

More recently, benchmarks such as SWE-bench [14] have confronted models with even more challenging tasks, such as fixing GitHub issues on repository level.

The previously created QA dataset on scientific workflow code consists of 200 manually-created QA pairs on 10 reference Nextflow workflows. The dataset includes various types of questions, such as multiple choice or boolean questions. In addition, the dataset contains labels on which workflow language elements are relevant to each QA pair. Also, supporting facts are annotated, i.e. the lines of code relevant to answering the respective question [1].

3 Goals

For the existing QA dataset on scientific workflow code, we observed that as context length increased for workflows with more lines of code, model performance dropped [1]. These findings suggest that even LLMs developed specifically for long contexts such as the Nemotron-Ultralong-1M model [31] struggle to identify relevant passages in long context documents.

For most questions in the dataset, only a small number of code snippets is necessary to provide a correct answer. Our goal is to reduce context length

by retrieving relevant code snippets from a reference workflow for a given question. Subsequently, only the retrieved code should be supplied to the model along with the question, sparing the model the task of long-range in-context retrieval. Such retrieval pipelines can enhance models output [33], as well as shrink inference cost [32].

4 Methodology

In order to correctly retrieve relevant code snippets for a given query, we aim to apply several retrieval techniques:

1. First, we seek to use a hybrid model combining sparse and dense retrieval, since such systems proved to improve retrieval results [25, 30].

2. Next, we intend to implement an iterative retrieval approach which uses previously retrieved items for the following step. This should be helpful for questions that require reasoning across multiple processes or subworkflows.

3. Third, abstractive compression could be used to target retrieved code snippets more specifically to the query (by removing irrelevant lines of code). Retrieved code snippets often still contain a significant amount of code that is not relevant to the given query - many QA pairs in the already existing dataset only need few lines to be answered correctly.

To implement the retriever described above, a development dataset (e.g. for hyperparameter optimization) is necessary. Since creating QA pairs manually is a time consuming task, we aim to apply a more scalable approach for extending the dataset. Question generation is the most labor-intensive step and could be handled by language models. In the past, QA datasets have been generated [40] or augmented [9] using LLMs. We aim to apply a few-shot approach: The model is presented with few examples when being prompted to generate new questions. Few-shot approaches have shown the ability to improve model output in general [2] and for generating datasets [41]. To ensure comparability to the manually-created dataset, question filtering and answer annotation are taken care of manually.

We seek to compare the developed retriever with baselines such as BM25, and also aim to conduct an ablation study, examining the impact of iterative retrieval and information compression (as described above) separately.

5 Evaluation

We retain the complete manually created QA dataset for evaluation, along with a subset of the synthetically generated questions. The dataset contains annotations on relevant information for each QA pairs: Supporting facts mark the lines of code that are most important to answer a given question. These annotations can be used to evaluate whether a system successfully retrieves the code snippets relevant to the question.

Next to evaluating the retriever system, we also aim to integrate the retriever in a RAG pipeline for QA, and evaluate changes in performance for LLMs compared to settings without an intermediary retriever.

References

[1] S. Bosse. Creating a question answering dataset for scientific workflow code. 2025.

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[3] J. Chen, K. Zhao, J. Liu, C. Peng, J. Liu, H. Zhu, P. Gao, P. Yang, and S. Deng. Coreqa: uncovering potentials of language models in code repository question answering. *arXiv preprint arXiv:2501.03447*, 2025.

[4] Y. Chen, T. Chen, H. Jhamtani, P. Xia, R. Shin, J. Eisner, and B. Van Durme. Learning to retrieve iteratively for in-context learning. *arXiv preprint arXiv:2406.14739*, 2024.

[5] W. Cheng, Y. Wu, and W. Hu. Dataflow-guided retrieval augmentation for repository-level code completion. *arXiv preprint arXiv:2405.19782*, 2024.

[6] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, and C. Notredame. Nextflow enables reproducible computational workflows. *Nature biotechnology*, 35(4):316–319, 2017.

[7] P. A. Ewels, A. Peltzer, S. Fillinger, H. Patel, J. Alneberg, A. Wilm, M. U. Garcia, P. Di Tommaso, and S. Nahnsen. The nf-core framework for community-curated bioinformatics pipelines. *Nature biotechnology*, 38(3):276–278, 2020.

[8] Y. Gu, J. Cao, S. Qian, and W. Guan. Sworts: a scientific workflow retrieval approach by learning textual and structural semantics. *IEEE Transactions on Services Computing*, 16(6):4205–4219, 2023.

[9] Z. Guo, P. Wang, Y. Wang, and S. Yu. Improving small language models on pubmedqa via generative data augmentation. *arXiv preprint arXiv:2305.07804*, 2023.

[10] K. A. Hambarde and H. Proenca. Information retrieval: recent advances and beyond. *IEEE Access*, 11:76581–76604, 2023.

[11] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 33(8):1–79, 2024.

[12] R. Hu, C. Peng, J. Ren, B. Jiang, X. Meng, Q. Wu, P. Gao, X. Wang, and C. Gao. Coderepoqa: A large-scale benchmark for software engineering question answering. *arXiv preprint arXiv:2412.14764*, 2024.

[13] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.

[14] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

[15] V. Karpukhin, B. Oguz, S. Min, P. S. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih. Dense passage retrieval for open-domain question answering. In *EMNLP (1)*, pages 6769–6781, 2020.

[16] J. Köster and S. Rahmann. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 2012.

[17] C. Lee, Y. Seonwoo, and A. Oh. Cs1qa: A dataset for assisting code-based question answering in an introductory programming course. *arXiv preprint arXiv:2210.14494*, 2022.

[18] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[19] L. Li, S. Geng, Z. Li, Y. He, H. Yu, Z. Hua, G. Ning, S. Wang, T. Xie, and H. Yang. Infibench: Evaluating the question-answering capabilities of code large language models. *Advances in Neural Information Processing Systems*, 37:128668–128698, 2024.

[20] X. Li, K. Dong, Y. Q. Lee, W. Xia, H. Zhang, X. Dai, Y. Wang, and R. Tang. Coir: A comprehensive benchmark for code information retrieval models. *arXiv preprint arXiv:2407.02883*, 2024.

[21] Z. Li, J. Zhang, C. Yin, Y. Ouyang, and W. Rong. Procqa: a large-scale community-based programming question answering dataset for code search. *arXiv preprint arXiv:2403.16702*, 2024.

[22] X. Ling, L. Wu, S. Wang, G. Pan, T. Ma, F. Xu, A. X. Liu, C. Wu, and S. Ji. Deep graph matching and searching for semantic code retrieval. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(5): 1–21, 2021.

[23] C. Liu and X. Wan. Codeqa: A question answering dataset for source code comprehension. *arXiv preprint arXiv:2109.08365*, 2021.

[24] T. Liu, C. Xu, and J. McAuley. Repobench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*, 2023.

[25] Y. Luan, J. Eisenstein, K. Toutanova, and M. Collins. Sparse, dense, and attentional representations for text retrieval. *Transactions of the Association for Computational Linguistics*, 9:329–345, 2021.

[26] X. Ma, Y. Gong, P. He, H. Zhao, and N. Duan. Query rewriting in retrieval-augmented large language models. In *Proceedings of the 2023*

Conference on Empirical Methods in Natural Language Processing, pages 5303–5315, 2023.

[27] N. Maligeay, N. Bossut, and K. Belhajjame. Why do scientific workflows still break? In *Proceedings of the 36th International Conference on Scientific and Statistical Database Management*, pages 1–4, 2024.

[28] J. Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 29–48. New Jersey, USA, 2003.

[29] J. Starlinger, B. Brancotte, S. Cohen-Boulakia, and U. Leser. Similarity search for scientific workflows. *Proceedings of the VLDB Endowment (PVLDB)*, 7(12):1143–1154, 2014.

[30] X. Wang, Z. Wang, X. Gao, F. Zhang, Y. Wu, Z. Xu, T. Shi, Z. Wang, S. Li, Q. Qian, et al. Searching for best practices in retrieval-augmented generation. *arXiv preprint arXiv:2407.01219*, 2024.

[31] C. Xu, W. Ping, P. Xu, Z. Liu, B. Wang, M. Shoeybi, B. Li, and B. Catanzaro. From 128k to 4m: Efficient training of ultra-long context large language models. *arXiv preprint arXiv:2504.06214*, 2025.

[32] F. Xu, W. Shi, and E. Choi. Recomp: Improving retrieval-augmented lms with compression and selective augmentation. *arXiv preprint arXiv:2310.04408*, 2023.

[33] P. Xu, W. Ping, X. Wu, L. McAfee, C. Zhu, Z. Liu, S. Subramanian, E. Bakhturina, M. Shoeybi, and B. Catanzaro. Retrieval meets long context large language models. *arXiv preprint arXiv:2310.03025*, 2023.

[34] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. W. Cohen, R. Salakhutdinov, and C. D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.

[35] A. B. Yoo, M. A. Jette, and M. Grondona. Slurm: Simple linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing*, pages 44–60. Springer, 2003.

[36] F. Zhang, B. Chen, Y. Zhang, J. Keung, J. Liu, D. Zan, Y. Mao, J.-G. Lou, and W. Chen. Repocoder: Repository-level code completion through iterative retrieval and generation. *arXiv preprint arXiv:2303.12570*, 2023.

[37] J. Zhang, H. Zhang, D. Zhang, Y. Liu, and S. Huang. End-to-end beam retrieval for multi-hop question answering. *arXiv preprint arXiv:2308.08973*, 2023.

[38] W. X. Zhao, J. Liu, R. Ren, and J.-R. Wen. Dense text retrieval based on pretrained language models: A survey. *ACM Transactions on Information Systems*, 42(4):1–60, 2024.

[39] Y. Zhu, H. Yuan, S. Wang, J. Liu, W. Liu, C. Deng, H. Chen, Z. Liu, Z. Dou, and J.-R. Wen. Large language models for information retrieval: A survey. *arXiv preprint arXiv:2308.07107*, 2023.

[40] Y. Zhuang, Y. Yu, K. Wang, H. Sun, and C. Zhang. Toolqa: A dataset for llm question answering with external tools. *Advances in Neural Information Processing Systems*, 36:50117–50143, 2023.

[41] I. Ziegler, A. Köksal, D. Elliott, and H. Schütze. Craft your dataset: Task-specific synthetic dataset generation through corpus retrieval and augmentation. *arXiv preprint arXiv:2409.02098*, 2024.