A Dense Retrieval System and Evaluation Dataset for Scientific Computational Notebooks

Na Li*, Yangjun Zhang†, Zhiming Zhao*‡
*Multiscale Networked System, Informatics Institute, University of Amsterdam, Netherlands
†School of Electronic Engineering and Computer Science, Queen Mary University of London, United Kingdom
‡LifeWatch ERIC Virtual Lab and Innovation Center (VLIC), Amsterdam, Netherlands
n.li@uva.nl, yangjun.zhang@qmul.ac.uk, z.zhao@uva.nl

***Abstract*—The discovery and reutilization of scientific codes are crucial in many research activities. Computational notebooks have emerged as a particularly effective medium for sharing and reusing scientific codes. Nevertheless, effectively locating relevant computational notebooks is a significant challenge. First, computational notebooks encompass multi-modal data comprising unstructured text, source code, and other media, posing complexities in representing such data for retrieval purposes. Second, the absence of evaluation datasets for the computational notebook search task hampers fair performance assessments within the research community. Prior studies have either treated computational notebook search as a code-snippet search problem or focused solely on content-based approaches for searching computational notebooks. To address the aforementioned difficulties, we present DeCNR, tackling the information needs of researchers in seeking computational notebooks. Our approach leverages a fused sparse-dense retrieval model to represent computational notebooks effectively. Additionally, we construct an evaluation dataset including actual scientific queries, computational notebooks, and relevance judgments for fair and objective performance assessment. Experimental results demonstrate that the proposed method surpasses baseline approaches in terms of *F1@5* and *NDCG@5*. The proposed system has been implemented as a web service shipped with REST APIs, allowing seamless integration with other applications and web services.**

***Index Terms*—computational notebook search, evaluation dataset, Jupyter Notebook, scientific code reuse, virtual research environment**

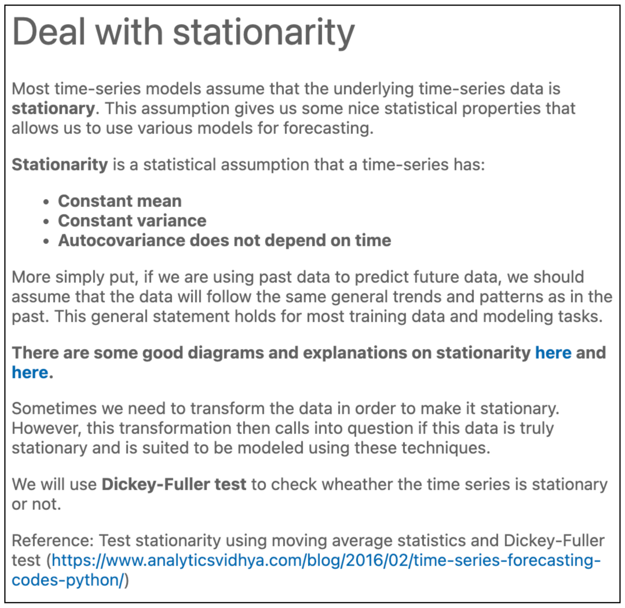
I. Introduction

In the rapidly evolving landscape of scientific research, scientists often need to find and reuse codes for various purposes, e.g., learn state-of-the-art algorithms, reproduce previous research results, and quickly develop new methods. Computational notebooks have emerged as a particularly effective medium for scientific code sharing and reusing [1]. The computational notebook programming environment allows researchers to interweave code with explanatory text, images, and other media, making it easier to communicate their thought processes and provide context for their codes [2]. Collecting computational notebooks and building a search system help scientists discover research resources published through computational notebooks and thus facilitate knowledge sharing and increase productivity [3].

There are several studies on computational notebook search [4]–[6]. A branch of work emphasizes code fragments (a reusable block of code, e.g., a code cell [4], or a sequence of codes extracted from different code cells [5]) of computational notebooks and returns code snippets given natural language queries. They treat computational notebook search as a code-snippet search problem and aim to bridge the gap between programming language and natural language. While code-snippet search solutions can be effective for general programming questions, they may not be suitable for answering research questions that involve complex pipelines or workflows. Others investigate content-based computational notebook search (or notebook-to-notebook search), which uses computational notebooks as queries to retrieve relevant computational notebooks [6]. These methods provide a representation for the entire computational notebook, e.g., source codes, tabular data, output formats, and imported library names, but limit the application to scenarios where users already have a computational notebook.

Despite studies carried out on computational notebook search, researchers still rely on general search engines or open-source code repositories to find computational notebooks. These tools do not explicitly index the contents of computational notebooks and may yield inaccurate results or take longer to locate the relevant ones. This leads to the goal of our work, i.e., building an effective computational notebook search system. However, two significant challenges need to be addressed. The first challenge is the need for suitable representations of both text and code in computational notebooks. Despite the availability of advanced models for processing natural language and programming languages, representing computational notebooks is still difficult. This is primarily because computational notebooks are multi-modal and need to cater to complex scientific information requirements. For example, in Figure 1, a Markdown cell and a code cell together provide knowledge on a specific topic, such as the stationarity test for time series data. The Markdown cell explains the concept of stationarity, while the code cell demonstrates a stationarity test on real data with visualization. The combination of text and code is common in computational notebooks due to their literate programming approach. The second challenge is the lack of evaluation datasets in the field of computational notebook search, making it difficult to assess and compare different methods for searching computational notebooks objectively and fairly.

Targeting the above challenges, we (1) propose Dense


Deal with stationarity

Most time-series models assume that the underlying time-series data is **stationary**. This assumption gives us some nice statistical properties that allows us to use various models for forecasting.

Stationarity is a statistical assumption that a time-series has:

- **Constant mean**
- **Constant variance**
- **Autocovariance does not depend on time**

More simply put, if we are using past data to predict future data, we should assume that the data will follow the same general trends and patterns as in the past. This general statement holds for most training data and modeling tasks.

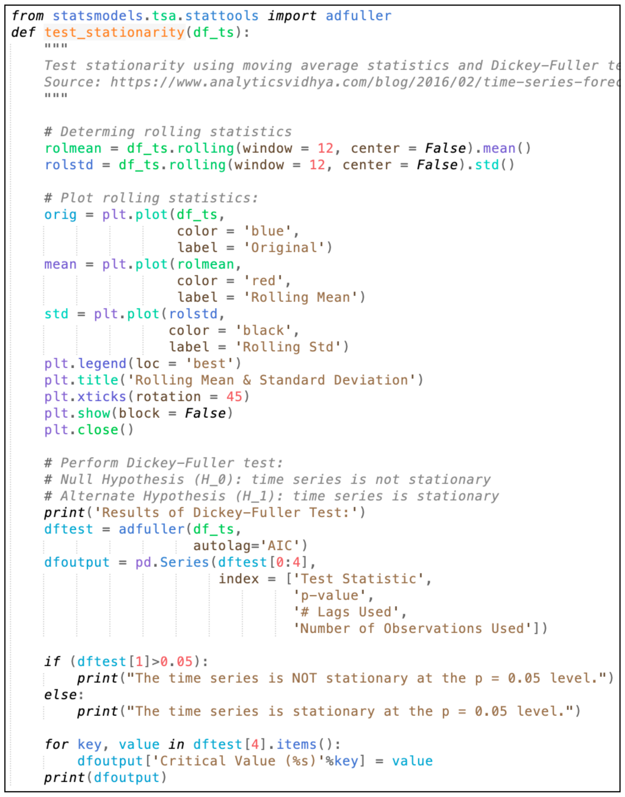
There are some good diagrams and explanations on stationarity here and here.

Sometimes we need to transform the data in order to make it stationary. However, this transformation then calls into question if this data is truly stationary and is suited to be modeled using these techniques.

We will use **Dickey-Fuller test** to check wheather the time series is stationary or not.

Reference: Test stationarity using moving average statistics and Dickey-Fuller test (https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/)

(a)


```
from statsmodels.tsa.stattools import adfuller
def test_stationarity(df_ts):
    """
    Test stationarity using moving average statistics and Dickey-Fuller te
    Source: https://www.analyticsvidhya.com/blog/2016/02/time-series-forec
    """

    # Determing rolling statistics
    rolmean = df_ts.rolling(window = 12, center = False).mean()
    rolstd = df_ts.rolling(window = 12, center = False).std()

    # Plot rolling statistics:
    orig = plt.plot(df_ts,
                    color = 'blue',
                    label = 'Original')
    mean = plt.plot(rolmean,
                    color = 'red',
                    label = 'Rolling Mean')
    std = plt.plot(rolstd,
                   color = 'black',
                   label = 'Rolling Std')
    plt.legend(loc = 'best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.xticks(rotation = 45)
    plt.show(block = False)
    plt.close()

    # Perform Dickey-Fuller test:
    # Null Hypothesis (H_0): time series is not stationary
    # Alternate Hypothesis (H_1): time series is stationary
    print('Results of Dickey-Fuller Test:')
    dftest = adfuller(df_ts,
                      autolag='AIC')
    dfoutput = pd.Series(dftest[0:4],
                         index = ['Test Statistic',
                                  'p-value',
                                  '# Lags Used',
                                  'Number of Observations Used'])

    if (dftest[1]>0.05):
        print("The time series is NOT stationary at the p = 0.05 level.")
    else:
        print("The time series is stationary at the p = 0.05 level.")

    for key, value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
```

(b)

Fig. 1. An example of a computational notebook. Source link: https://github.com/MRYingLEE/Time-series-Preprocessing-Studio-in-Jupyter/blob/master/Time-seriesPreprocessing.ipynb

Computational Notebook Retrieval (DeCNR), which models computational notebooks as *bi-modal* data (including text and code) and utilizes a fused sparse-dense model for computational notebooks retrieval; (2) build an evaluation dataset, including actual scientific queries, computational notebooks, and relevance judgments for performance assessment. The related artifacts can be found here: https://github.com/nali001/notebook_search_docker.

II. Related work

This section reviews previous studies on computational notebook search/retrieval that are highly relevant to our work.

A. Computational Notebook Retrieval

Research on notebook search is still in its infancy, with only a handful of prior studies [4]–[6]. Previous research can be summarized into two categories: *code search in computational notebooks* and *content-based computational notebook search*. NBSearch [4] and EDAssistant [5] support semantic code search in a computational notebook collection. NBSearch first translates source codes into textual descriptors and then computes the similarity between the queries and the descriptors using language models [4]. EDAssistant focuses on the Exploratory Data Analysis (EDA) sequence of computational notebooks and utilizes GraphCodeBERT [7] to represent the codes. Both are aimed at code snippet search and lack the representation approach for the entire computational notebook. JupySim [6] is a content-based notebook search system. The query comprises codes in cells, tabular data, output formats, and libraries. The similarity between queries and notebooks is defined as the weighted sum of similarities for each type of content. However, there is no clear explanation of the similarity computation methods for separated contents. Moreover, it is hard to evaluate the proposed system's efficiency and usability since no experimental results are reported.

Different from previous studies that search code blocks from computational notebooks or use computational notebooks as queries, we propose searching for the entire computational notebooks using natural-language queries.

B. Text and Code Representation for Computational Notebooks

Computational notebook representation is the cornerstone of computational notebook search systems. Computational notebooks comprise texts and codes, each of which can be represented effectively for retrieval. Related studies can be summarized into two categories: text representation and code representation. There is an enormous body of studies dedicated to text representation. Classical text retrieval methods, e.g., BM25 [8], use term-level heuristic statistics to represent the documents. Despite its simplicity, it is still effective in many retrieval tasks [9]. However, they only consider the lexical retrieval similarity between queries and documents and thus fail to retrieve relevant documents when there are no overlapping words in the queries and documents. Neural retrieval models are then developed for semantic retrieval [10]. Among them, dense retrieval methods [11] based on large-scale pre-trained language models [12], [13], e.g., DPR [14] and SBERT [15], map both query and document into a continuous vector space (dense vector space) where semantically similar words, phrases and sentences are closer to each other, and thus also called semantic retrieval methods. Similarly, two types of code modeling methods exist: Information Retrieval (IR)-based and Machine Learning (ML)-based methods. IR-based code retrieval methods [16], e.g., codehow [17], utilize traditional text retrieval techniques to measure text similarity and rely on shared words between queries and codes. ML-based methods usually map text and code snippets into a shared vector space. The relevance can be measured through cosine similarity between vector representations [18]–[22].

Unlike previous studies focusing on the representation of a single modality of data, i.e., text or code, we consider both text and code to represent computational notebooks.

C. Hybrid Sparse-dense Retrieval

Sparse retrieval models [8] usually utilize the statistical characteristics of words to represent queries and computational notebooks. Dense retrieval models [11] aim to map texts and codes into a continuous vector space, and the similarity between queries and computational notebooks can be computed as the dot product between the vector representations. Each type of method has pros and cons. For instance, sparse retrieval models are highly efficient but limited by their lexical essence. Dense retrieval models provide preferable semantic matching between words and sentences but usually require in-domain labeled data for training or fine-tuning. We propose a fusion-based approach to overcome this issue, combining sparse and dense retrieval models to derive the final ranking. Hybrid retrieval methods have been studied to improve the effectiveness of a retrieval system via the fusion of different retrieval strategies [23]–[25]. Chen et al. [25] propose a simple yet effective zero-shot hybrid model that combines BM25 with NPR [26] to address the out-of-domain generalization problem. They also point out that the dense retrieval models could be incompetent in modeling long documents, and sparse retrieval models can compensate for these weaknesses. Computational notebooks and scientific queries are out-of-domain data for most large-scale pre-trained language models, e.g., SBERT [15] and computational notebook retrieval also faces the long document problem. Therefore, the combination of sparse and dense models can be a potential solution for out-of-domain and long document problems in computational notebook retrieval. Hence, different from the previous studies [4], [5] that use only one type of model, we propose a fused sparse-dense model for computational notebook retrieval.

III. Dense Computational Notebook Retrieval

This section describes the proposed DeCNR, a system that enables searching for external computational notebooks to support scientific research activities.

A. System Design

Figure 2 (a) depicts the high-level architecture of the system. We harvest raw computational notebook and their corresponding metadata from the web through a *Crawler*. The system then processes each data accordingly. The computational notebooks first go through a *Preprocessor* to extract useful contents and then input to a *Computational Notebook Representation* module. Meanwhile, the metadata for each computational notebook is handled by a *Metadata Harmonization* unit to generate common metadata. The *Indexer* proceeds with preprocessed contents or representations to generate indexes to be stored in the *Index Database*. The *Retriever* will serve online retrieval for computational notebooks. More details of each module are provided in the following:

1) A *Crawler* collects computational notebooks and corresponding metadata from the web.
2) A *Preprocessor* takes raw contents of computational notebooks as input and outputs aggregated texts and codes.
3) A *Computational Notebook Representation* unit transforms natural-language texts and codes to vectors.
4) A *Metadata Harmonization* unit maps metadata from various data sources onto a common metadata schema to ease the indexing for the metadata.
5) An *Indexer* generates indexes from vector representations produced by the *Computational Notebook Representation* unit and common metadata produced by the *Metadata Harmonization* unit.
6) An *Index Database* stores the indexes and makes them available for searching.
7) A *Retriever* interacts with users via the *User Interface*, and retrieves matched notebooks from the *Index Database* based on their similarities.

The system can be used to handle computational notebooks from different sources. Typical data sources include software repositories (e.g., GitHub and Kaggle), research assets catalogs, or other public web locations. Since crawling computational notebooks is a common practice, our work only focuses on the rest of the pipeline. The user interface can be dedicated web pages for vertical computational notebook search or extensions of existing web services. The backend can be scaled to multiple instances and run on cloud infrastructure. These components ' technical considerations and detailed design will be discussed in the rest of this section.

B. Preprocessor

The preprocessor is responsible for splitting texts from codes and cleaning up irrelevant content such as images and cell outputs. A computational notebook is in JSON format and is organized as cells. We traverse the cells to extract contents inside, concatenate texts from all Markdown cells, and codes from all code cells. Afterwards, we omit the HTML contents, URLs, and blank lines inside the text to get clean textual descriptions. For code, we first remove the commands commonly used in Jupyter Notebooks but not part of the code itself, e.g., cell magic or line magic commands, and shell commands to install Python packages or set environment variables. Next, we enhance the metadata by incorporating statistical features, such as the count of code cells.

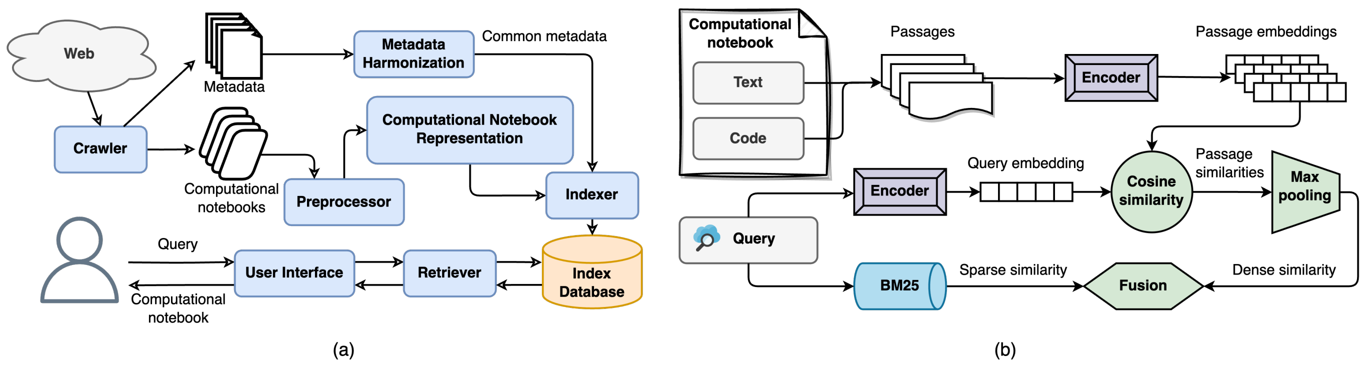


Fig. 2. (a) The high-level structure of DeCNR; (b) Computational notebook ranking using the fusion of BM25 and a dense retrieval model.

C. Indexer, Index Database and Retriever

The indexer, together with the index database, is necessary for the fast retrieval of computational notebooks. Inverted indexing is commonly exploited to allow fast full-text searches based on sparse retrieval models, e.g., BM25 and TF-IDF. Recently, tools that support fast similarity search of dense vectors, e.g., Faiss [27], have emerged as the backbone of dense retrieval systems. In our case, we use Elasticsearch[1] as the indexer to serve the “shallow” retrieval of computational notebooks, which are based on the lexical similarity between queries and the textual description of computational notebooks. To support “semantic” retrieval, we use haystack [28] to generate Faiss indexes with dense retrieval methods. The retriever interacts with users and the index database. It receives user queries, retrieves notebooks, and ranks them based on their similarities to the queries.

D. Computational Notebook Ranking using a Fused Model

We use a fused sparse-dense model for computational notebook retrieval. Figure 2 (b) illustrates the workflow of the proposed method. A computational notebook is first split into ‘text’ and ‘code’, with ‘text’ being aggregated textual descriptions in Markdown cells and ‘code’ the code fragments from code cells. The texts and codes are concatenated and then segmented into passages to fit in the input layer of the *Encoder*. Each passage p is encoded as a vector representation in a high-dimensional Euclidean space, referred to as a *Passage embedding* $v_p = E_P(p)$, where E_P is the encoder for passages. Another encoder E_Q encodes a query q to generate a query vector representation $v_q = E_Q(q)$. The *Cosine similarity* between the passage and the query is computed as the inner product of two vector representations, i.e., $v_p \cdot v_q$. We apply the *Max-pooling* operation to all passage similarities to get the computational notebook-level similarity score. Meanwhile, A sparse model computes the similarity score between the query and the computational notebook. For sparse and dense model fusion, we employ the linear combination of similarity scores computed by two models. Formally, for each query and computational notebook pair (q, d) the final similarity score S_{fusion} is derived by:

$$S_{fusion} = w_1 * S_{dense} + w_2 * S_{sparse}, \tag{1}$$

where S_{dense} and S_{sparse} refer to similarity scores produced by the dense and the sparse retrieval model, respectively. w_1 and w_2 are weighting parameters. The computational notebooks are re-ranked by the fused similarity score S_{fusion}.

In the current implementation, we use BM25 as the sparse retrieval model and SBERT [15] as the dense retrieval model. SBERT utilizes pre-trained large language models [12], [13] and fine-tunes them to achieve semantically meaningful vector representations of sentences for efficient similarity search.

E. Metadata Harmonization

Computational notebooks come with default metadata such as “name”, “full_name”, “id” and “language” when downloaded from big code repositories, such as Kaggle and GitHub. However, they usually follow different schemas and pose difficulties in metadata indexing. Therefore, we introduce a metadata harmonization unit that extracts and maps the most common fields from different metadata schemas, which are the above rows (until the description) displayed in Table I. These metadata fields provide the index’s basis but are insufficient for understanding the contents. The pipeline extracts other meta-information directly from the computational notebook contents. We add six main fields (the bottom rows in Table I) to store additional features. The computational notebooks from different sources are indistinguishable from each other in terms of the contents, so we append the same features to all computational notebooks.

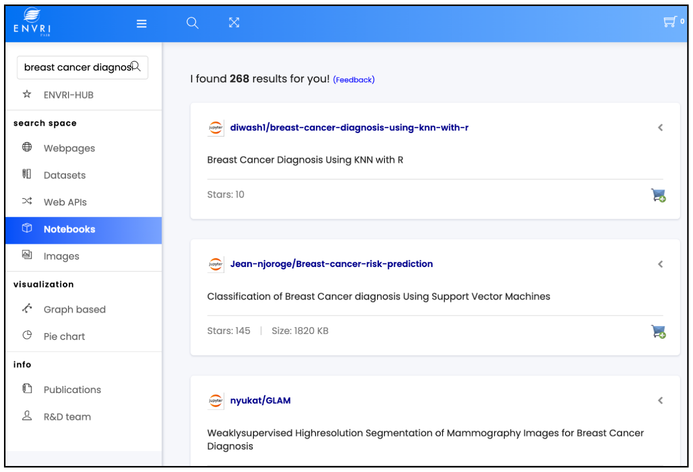
F. Implementation

The current software is implemented in the context of several EU projects, i.e., H2020 ENVRI-FAIR, CLARIFY, and BlueCloud. Computational notebooks are gathered from GitHub, Kaggle, and service catalogs provided by the data infrastructure in these projects. We emphasize extensibility, modularity, and deployability when developing the software. To cope with the rapid evolution of data analysis models, the backend is implemented with Python using the Django

[1]https://github.com/elastic/elasticsearch

TABLE I
THE SCHEMA OF HARMONIZED METADATA.

Mapped fields	Type	Explanation
docid	string	Local unique ID
stargazers_count	integer	Star numbers of the repository (GitHub only)
forks_count	integer	Fork numbers of the repository (GitHub only)
size	integer	File size (GitHub only)
name	string	The name of the notebook (Kaggle only)
html_url	string	URL of the computational notebook file
source	string	The name of the data source, e.g., GitHub
code_file	string	The name of the file
description	string	Text descriptions extracted from Markdown cells
language	string	Programming language
num_cells	integer	Number of cells
num_code_cells	integer	Number of code cells
num_md_cells	integer	Number of Markdown cells
len_md_text	integer	Number of lines of texts inside Markdown cells



I found **268** results for you! (Feedback)

diwash1/breast-cancer-diagnosis-using-knn-with-r

Breast Cancer Diagnosis Using KNN with R

Stars: 10

Jean-njoroge/Breast-cancer-risk-prediction

Classification of Breast Cancer diagnosis Using Support Vector Machines

Stars: 145 | Size: 1820 KB

nyukat/GLAM

Weaklysupervised Highresolution Segmentation of Mammography Images for Breast Cancer Diagnosis

Fig. 3. DeCNR embedded with the ENVRI search platform.

framework, considering that many SOTA retrieval models are available through Python libraries, e.g., transformers, and thus can be smoothly updated. To modularize the system, we align the software components with conceptual modules demonstrated in Figure 2 and separate them into self-contained units. This is achieved by the careful design of the communication interface between two connected modules. Regarding the deployment, we leverage Docker's containerization techniques to ease the deployment of the system. The system can be effortlessly deployed in a single laptop, a virtual machine, or a cluster.

Additionally, we developed a search agent as a Jupyter extension tool inside our NaaVRE [29] ecosystem to demonstrate the in-site usage of computational notebook search within a Jupyter-based Virtual Research Environment. It is a key supporting element in achieving the vision of "Notebook as a Virtual Research Environment" [29]. The user interface is the same as that in CNSVRE [30]. We also incorporate it into the ENVRI[2] search platform, displaying search results on the website, as shown in Figure 3. In this case, the search is only performed over metadata due to practical reasons. Additionally, it provides REST APIs that can be incorporated into any web service.

IV. Evaluation

To assess the performance of proposed computational notebook search methods and facilitate future studies, we build an evaluation dataset containing scientific queries, computational notebooks, and relevance judgments. This section describes the evaluation dataset construction procedure.

A. Query Collection

We collect researchers' queries by designing and circulating a questionnaire within the university and the network of Ph.D. students in the related EU projects. The questionnaire explicitly instructs the participants to provide research-related code search queries. More concretely, they are asked to "list at least 5 queries in natural language that you use for searching codes related to your research topic(s). The queries should reflect your research topics/questions, e.g., segmentation of epidermis in histopathological images, instead of general coding tips, e.g., load json files". Additionally to the queries, we also ask them to specify their research fields. A total of 132 queries are collected from 26 participants, with most of them from the computer science domain and two from the chemistry and psychology domains. Table II lists some examples of the collected queries.

[2]https://search.envri.eu/genericpages/

B. Relevance Judgment Collection

Finding relevant computational notebooks given an arbitrary query is a significant challenge using current search platforms. Thus, we adopt the following steps to generate relevance judgments for the queries and the computational notebooks, shown in Figure 4. We collect computational notebooks from GitHub, a commonly used code repository where computational notebooks are stored within repositories. We use APIs to search and download computational notebooks. Since the returned results are scarce for most original queries, we first enlarge the query set by manually extracting scientific entities (words or phrases standing for scientific concepts, e.g., attention-based multiple instance learning, pytorch, whole slide image diagnosis) from the original queries and combine these entities into new queries, resulting in 270 queries. We then use the enlarged query set as keywords to search GitHub at the repository level because searching at the file level produces too many noisy results. The constraint of the "Jupyter Notebook" language is applied to pick out repositories containing computational notebooks. We selected 64 queries that have 1-100 returned code repositories and downloaded all the *.ipynb* files from the top 30 code repositories, ranked by the number of stars. The reason behind using 1-100 repositories as selection criteria is that queries that result in more than 100 repositories are usually too broad, which will not be well supported by the search system. For relevance judgment, we use the original queries collected from researchers instead of the processed

TABLE II

QUERIES COLLECTED FROM RESEARCHERS WITH DIVERSE BACKGROUNDS

Index	Query	Research field
1	Post processing in gigapixel images	Image Processing
2	Multi-class tissue segmentation with Pytorch	artificial intelligence for histopathological image analysis
3	multi-modal deep learning models for early detection of AD	the early detection of Alzheimer's disease
4	Specularity removal from image	Intrinsic Image Decomposition, Augmented Reality & Colour Constancy
5	Time-series data preprocessing	Cloud computing
6	Internet of things	Computer science
7	Comparison of execution cost in smart contracts	Blockchain
8	code for t test	Psychology
9	codes for calculating photoluminescence quantum yield (PLQY)	Chemistry

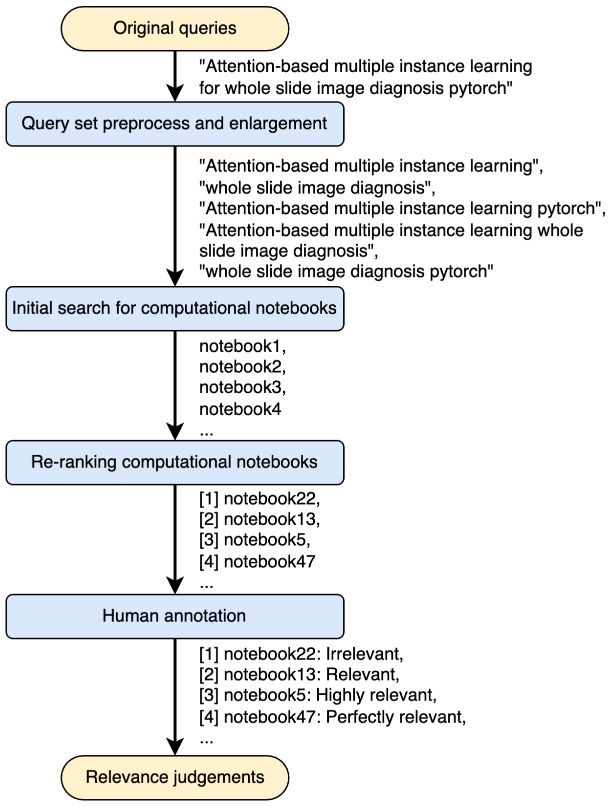

Original queries
"Attention-based multiple instance learning for whole slide image diagnosis pytorch"
Query set preprocess and enlargement
"Attention-based multiple instance learning",
"whole slide image diagnosis",
"Attention-based multiple instance learning pytorch",
"Attention-based multiple instance learning whole slide image diagnosis",
"whole slide image diagnosis pytorch"
Initial search for computational notebooks
notebook1,
notebook2,
notebook3,
notebook4
...
Re-ranking computational notebooks
[1] notebook22,
[2] notebook13,
[3] notebook5,
[4] notebook47
...
Human annotation
[1] notebook22: Irrelevant,
[2] notebook13: Relevant,
[3] notebook5: Highly relevant,
[4] notebook47: Perfectly relevant,
...
Relevance judgements

Fig. 4. Steps for collecting relevance judgments between queries and computational notebooks.

queries used for computational notebook collection. Thus, the 64 queries are projected to 47 original queries, which will be included in the evaluation dataset.

It is infeasible to assess the relevance of queries against all computational notebooks. Therefore, we use the BM25 method to re-rank the collected computational notebooks for each query to present more relevant ones in the top positions. Afterward, we manually annotate the top-ranked computational notebooks in terms of relevance to the given query. To ensure at least one relevant record for each query, relevant computational notebooks are quested using multiple search platforms and added into the search space. We utilize four-level relevance labels for annotation, similar to that in the MS Marco dataset [31]. Eventually, the evaluation dataset contains 49 queries, 3,779 computational notebooks, and 254 relevance judgments. Table III summarizes the statistics.

V. EXPERIMENTS

We aim to answer the following research questions via empirical experiments: **RQ1** What contents within the computational notebooks, including texts and codes, are useful for retrieval? **RQ2** How do dense retrieval models perform, compared with sparse retrieval models? **RQ3** How does model fusion affect the performances of dense retrieval models?

A. Experimental Setup

1) Query subjects: To examine the ability of our system to support diversified scientific queries, we use the 47 queries from the evaluation dataset, as described in Section IV-B.

2) Search space: The search space used for evaluating our system comprises the 3,779 computational notebooks crawled from GitHub using the collected queries. All queries are accompanied by at least one relevant computational notebook. The relevance labels are in four scales: {*3–Perfectly relevant, 2–Highly relevant, 1–Relevant, 0–Irrelevant*}. Computational notebooks are split into passages of 512 words. The number of passages for different types of contents is listed in Table III.

3) Evaluation metrics: We use *precision*, *recall*, *F1*, and *Normalized Discounted Cumulative Gain (NDCG)* as the performance metrics, which are commonly used in information retrieval tasks. Since users usually only pay attention to top-ranked results, we compute the metrics using top-k returned computational notebooks. The $precision@k$ measures the correct hits in the top-k ranking, and it is calculated as follows:

$$precision@k = \frac{TP@k}{k}, \tag{2}$$

where k is the number of retrieved computational notebooks, and $TP@k$ is the number of relevant ones in top-k retrieved results. The higher the $precision@k$ is, the more relevant results will be presented to users. The $recall@k$ measures how many relevant computational notebooks can be retrieved from the corpus:

$$recall@k = \frac{TP@k}{P}, \tag{3}$$

TABLE III
STATISTICS OF EVALUATION DATASET

# queries	# computational notebooks	# relevance judgments				# passages		
		Total	Avg.	Min.	Max.	Only text	Only code	Text and code
47	3,779	254 (208 relevant)	5.4	1	19	4,970	7,129	9,766

where P is the number of all relevant computational notebooks in the corpus. The $F1@k$ is the combination of $precision@k$ and $recall@k$, as defined below:

$$F1@k = 2 \times \frac{precision@k \times recall@k}{precision@k + recall@k}. \tag{4}$$

Note that the $precision@k$ is bounded by $\min(\frac{P}{k}, 1)$. In other words, the perfect precision 1 can not be achieved when $P < k$. Similarly, the $recall@k$ is limited to $\min(\frac{k}{P}, 1)$.

Despite the informativeness of the above metrics, they do not consider the ranking within the top-k results. For example, two rankings $[1, 1, 1, 0, 0, 0]$ and $[0, 0, 0, 1, 1, 1]$, where 1 denotes relevant and 0 non-relevant, will have the same values using these metrics. NDCG is commonly used to assess different ranking outputs, favoring relevant results at higher ranking positions. It is defined as below:

$$NDCG@k = \frac{DCG@k}{IDCG@k}, \tag{5}$$

where $DCG@k$ is the discounted cumulative gain at rank k and $IDCG@k$ is the ideal discounted cumulative gain at rank k. They are calculated as follows:

$$DCG@k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(i+1)}, \tag{6}$$

$$IDCG@k = \sum_{i=1}^{min(k,|N_a|)} \frac{2^{rel_{(i)}} - 1}{\log_2(i+1)}, \tag{7}$$

where rel_i is the relevance score of the i-th ranked computational notebook, while $rel_{(i)}$ is the relevance score of the i-th most relevant computational notebook in the corpus. $|N_a|$ is the size of the corpus.

Since the four-point labels are aimed at guiding human annotators in relevant judgment rather than providing accurate classification for the relevance of computational notebooks, we use binary labels when computing $precision@k$, $recall@k$, and $F1@k$. We regard the *Irrelevant* label as non-relevant and the rest as relevant. The four-point relevance labels remain for the $NDCG@k$ calculation. We report the model performances at k = 5 and 10. k=1 is not considered because including relevant computational notebooks in the top 5 or 10 results is sufficient for system usability.

4) Comparison methods: Two main factors affect the performance of computational notebook search methods: indexing contents and ranking models. Regarding indexing contents, we consider three settings: 'text', 'code', and 'text+code', which denote indexing only textual descriptions in the Markdown cells, only codes inside the code cells, and the concatenation of both. For the ranking models, we utilize a commonly used sparse retrieval model, BM25, and two state-of-art dense retrieval models: *sentence-transformers/multi-qa-mpnet-base-dot-v1* (SBERT1) and *sentence-transformers/all-mpnet-base-v2* (SBERT2). They are both based on the pre-trained model *microsoft/mpnet-base*[3] [13]. SBERT1 is trained on 215M (question, answer) pairs from diverse sources while SBERT2 is trained on above 1B sentence pairs. They share the same model size and report comparable performances on 6 diverse semantic search tasks. The dense retrieval models are provided by the sentence-transformers library[4] and are used in a zero-shot manner. All ranking methods are applied to the three aforementioned indexing settings. Besides single models, we investigate hybrid search via model fusion. Three fusion variants are considered: SBERT1+SBERT2, BM25+SBERT1, and BM25+SBERT2. The fused models utilize both texts and codes for similarity computation. We tested different settings of the weighting parameters w_1 and w_2 in the preliminary experiments, and the fused models seemed to be insensitive to these parameters. For simplicity, we set them both to 1. Their performances are measured using top-k ranked results.

B. Results

Table IV and Table V list the experimental results. As discussed in Section V-A3, the P@k and R@k have upper bounds depending on the selection of k. Therefore, we provide the metrics values in an ideal situation as references.

1) Effect of indexing contents (RQ1): We show the results in Table IV to demonstrate the effect of using different contents for retrieval. The results suggest that the combination of text and code performs better than only text or code. Moreover, the text is more useful than code for retrieving computational notebooks. First, we observe that the 'text+code' group consistently achieves the best performance using different retrieval models (Lines 3, 6, 9). Compared with the 'text' group, the P@5, R@5, and F1@5 increase by 45.2%, 47.7%, and 45.8% respectively using the BM25 method (Line 1, 3). Compared with the 'code' group, the P@5, R@5, and F1@5 increase by 281.2%, 136.1%, and 258.7% respectively using the BM25 method (Line 2, 3). Second, we find that the 'text' group has better performance than the 'code' group. Comparing the 'text' group to the 'code' group, the P@5, R@5, and F1@5 increase by 162.4%, 127.6%, and 142.2% respectively using the BM25 method (Line 1, 2). It is reasonable as, in many cases, texts and codes are complementary, and the combination of them provides more retrievable information for the computational

[3]https://huggingface.co/microsoft/mpnet-base
[4]https://www.sbert.net/

TABLE IV
Performance of models with different indexing contents. P@k: $precision@k$; R@k: $recall@k$, $k = 5, 10$. SBERT1: sentence-transformers/multi-qa-mpnet-base-dot-v1; SBERT2: sentence-transformers/all-mpnet-base-v2. 'text' stands for using only textual contents for indexing, 'code' for only codes, 'text+code' for the concatenation of texts and codes. Underline marks the best results within the same content group.

No.	Method	Content	P@5	R@5	F1@5	NDCG@5	P@10	R@10	F1@10	NDCG@10
0	–	Ideal	0.6851	0.9046	0.7073	–	0.4191	0.9856	0.5356	–
1		text	0.1787	0.2317	0.1843	0.4881	0.1426	0.3530	0.1872	0.5026
2	BM25	code	0.0681	0.1018	0.0761	0.2196	0.0553	0.1392	0.0742	0.2403
3		text+code	0.2596	0.3422	0.2730	0.5584	0.1915	0.5078	0.2593	0.5720
4		text	0.1787	0.2714	0.1927	0.4462	0.1298	0.3659	0.1761	0.4686
5	SBERT1	code	0.0936	0.1248	0.0951	0.2499	0.0660	0.1480	0.0802	0.2674
6		text+code	0.1872	0.2755	0.2021	0.4965	0.1426	0.3961	0.1948	0.5287
7		text	0.2043	0.3020	0.2217	0.5569	0.1426	0.3855	0.1909	0.5629
8	SBERT2	code	0.0723	0.1021	0.0761	0.1967	0.0660	0.1482	0.0829	0.2388
9		text+code	0.2213	0.3153	0.2356	0.5958	0.1553	0.4080	0.2084	0.5764

notebooks. Hence, both text and code should be exploited to better represent the computational notebooks.

2) Dense retrieval model performance (RQ2): We further investigate the effectiveness of dense retrieval models compared to the BM25 method using controlled indexing contents and report the results in Table IV. The results suggest that when using uni-modal data for indexing, i.e., text or code, dense retrieval models consistently outperform BM25. But BM25 tends to work better when using bi-modal data.

On the one hand, two dense retrieval models beat BM25 on all reported metrics under the 'text' and 'code' groups. Under the 'text' group, compared with BM25, SBERT2 sees increases of 14.3%, 30.3%, and 20.3% in terms of P@5, R@5, and F1@5 respectively (Line 1, 7). Under the 'code' group, compared with BM25, SBERT1 sees increases of 37.4%, 22.6%, and 25.0% in terms of P@5, R@5, and F1@5 respectively (Line 2, 5). The improvement of dense retrieval models compared with the BM25 method likely stems from the models' semantic representation of input data, which maps semantically similar sentences to closer places in a high-dimensional vector space, whereas BM25 matches words in a lexical manner. On the other hand, under the 'text+code' group, BM25 surpasses dense retrieval models w.r.t. P@k, R@k and F1@k, $k = 5, 10$. It implies that dense retrieval models are effective on single data modality, but applying them in computational notebook search tasks requires extra effort.

3) Effectiveness of the fusion method (RQ3): To examine the effectiveness of the fusion strategy and the effect of model selection for retrieval performance, we compare three variants of fused models, i.e., SBERT1+SBERT2, BM25+SBERT1, and BM25+SBERT2, applied on 'text+code' contents. The first method combines two dense retrieval models, while the last two coalesce a sparse retrieval model with a dense retrieval model. We show the results in Table V. They illustrate that incorporating a sparse retrieval model to form a keyword-aware retrieval method can significantly boost the performance of dense retrieval models. However, the fusion of two dense retrieval models does not improve and even impairs the performance compared to using a single dense retrieval model. The sparse-dense fused model gains the best performance for top-5 retrieved results.

First, both dense models achieve large performance increases when combined with the BM25 model. Compared with SBERT1 only, BM25+SBERT1 shows 42.1% and 12.5% raises on F1@k, for $k = 5, 10$ respectively (Line 2, 5). Compared with SBERT2 only, BM25+SBERT2 also manifests a performance improvement of 13.8% on F1@5 and 9.3% on F1@10 (Line 3, 6). In contrast, the SBERT1+SBERT2 combination causes a performance drop compared with each individual model (Line 2, 3, 4). Second, BM25+SBERT1 is the best-performed model over the top-5 retrieved computational notebooks. Compared with BM25, the P@5, R@5, F1@5 and NDCG@5 increase by 4.9%, 9.0%, 5.2% and 9.3%, respectively (Line 1, 5). An exception is P@10, R@10 and F1@10, the reason why the scores are low in metrics@10 could be that dense retrieval models introduce more irrelevant computational notebooks in the top-10 results. It provides the community with an opportunity to develop more effective fusion strategies for sparse and dense retrieval models to leverage the full set advantages of both models. Table VI shows top-3 ranking examples for the query "visual saliency" with BM25(text+code) and BM25+SBERT1(text+code) methods. Due to the page limit, we only display descriptive texts. Readers can access the whole computational notebooks via given URLs. Compared with BM25, the fused model of BM25 and SBERT1 improves the ranking by shifting relevant results one spot higher. Although it does not change the precision and recall for top-3 retrieved results, it prevents the frustration of seeing irrelevant computational notebooks on the top.

One common phenomenon in the experimental results is the decreased performance of P@k when k increases from 5 to 10, also seen in the ideal situation (Line 0 in Table IV). This is because the average number of relevant computational notebooks in the evaluation dataset is $208/47 \approx 4.4 < 5$, and

TABLE V
COMPARISON OF DIFFERENT METHODS USING 'TEXT+CODE' INDEXING CONTENTS.

No.	Method	P@5	R@5	F1@5	NDCG@5	P@10	R@10	F1@10	NDCG@10
1	BM25	0.2596	0.3422	0.2730	0.5584	**0.1915**	**0.5078**	**0.2593**	0.5720
2	SBERT1	0.1872	0.2755	0.2021	0.4965	0.1426	0.3961	0.1948	0.5287
3	SBERT2	0.2213	0.3153	0.2356	0.5958	0.1553	0.4080	0.2084	0.5764
4	SBERT1+SBERT2	0.1787	0.2668	0.1946	0.4954	0.1319	0.3834	0.1818	0.5368
5	BM25+SBERT1	**0.2723**	**0.3729**	**0.2872**	**0.6102**	0.1660	0.4234	0.2191	**0.6105**
6	BM25+SBERT2	0.2553	0.3381	0.2682	0.5808	0.1723	0.4241	0.2277	0.5754

TABLE VI
EXAMPLES OF RETURNED COMPUTATIONAL NOTEBOOKS FOR QUERY "VISUAL SALIENCY SALIENCY"

Method	Rank	Top-k ranked computational notebook	relevance
BM25 (text+code)	1	*DeepGaze II and ICF* https://github.com/sammy-w/visual_saliency/blob/main/deep_gaze/Demo.ipynb This notebook demonstrates how to load and use the DeepGaze II and ICF models. ...	0
	2	*Visualizing neural network with visual_keras* https://github.com/ellolo/visual-keras/blob/master/example_usage.ipynb This notebook provides examples of how to use the _visual__keras_ library to visualize neural network ...	2
	3	*Seeing is believing* https://github.com/davidGCR/VisualizeSaliency/blob/master/flashtorch-master/presentations/Hopperx1London/notebook.ipynb Using FlashTorch to shine a light on what neural nets "see" ...	3
BM25+SBERT1 (text+code)	1	*Visualizing neural network with visual_keras* https://github.com/ellolo/visual-keras/blob/master/example_usage.ipynb This notebook provides examples of how to use the _visual__keras_ library to visualize neural network ...	2
	2	*Seeing is believing* https://github.com/davidGCR/VisualizeSaliency/blob/master/flashtorch-master/presentations/Hopperx1London/notebook.ipynb Using FlashTorch to shine a light on what neural nets "see" Evolved in response to a desire to make neural nets ...	3
	3	*Find visual saliency in uncompressed videos* https://github.com/Dimitri78000/Neural_network_saliency/blob/master/Keras_saillancy.ipynb Dimitri LEURS and Khalil GHETARI framed by mihai MITREA. ...	0

thus the increase in the number of retrieved results will likely hamper the model precision. This also causes a prevalently higher R@k than P@k because the precision is more bounded than the recall by the small number of relevant computational notebooks associated with evaluation queries.

VI. CONCLUSION AND FUTURE WORK

With the Jupyter Notebook environment widely adopted in data science, there is an increasing amount of computational notebooks created and published on the web [32]. Scientists can reuse these resources to reduce laborious work and expedite scientific innovations. However, finding relevant computational notebooks is challenging due to the complexity of researchers' information needs and the multi-modal nature of computational notebooks. In this paper, we propose DeCNR to increase the discoverability of external computational notebooks via a fusion-based approach that combines BM25 and SBERT for computational notebooks ranking. Experimental results suggest that the proposed system can effectively retrieve semantically relevant computational notebooks, and the fusion-based model outperforms baseline models when evaluated on top-5 retrieved results.

Our bi-modal approach–considering both text and code within computational notebooks–addresses the inherent complexity and interplay of these two components. Notably, this approach has the potential to be extended beyond computational notebooks to encompass broader forms of data, such as code scripts and documentation. The proposed computational notebook search system is a great effort to improve the research environment, which relies heavily on commercial tools and open-domain retrieval techniques. Experimental results suggest that the proposed system can effectively retrieve semantically relevant computational notebooks, and the fused model outperforms baseline models when evaluated on top-5 retrieved results. Besides, the dataset developed for assessing computational notebook retrieval models fills an important gap in the research community. It enables convenient comparison between different methods and tracking of development in the field of computational notebook retrieval.

Nevertheless, we admit that there are still limitations in this work. We apply the dense retrieval models in a zero-shot manner, which may be a suboptimal solution towards the computational notebook search problem. Optimizing such models usually requires a large amount of labeled data, posing a significant challenge for research-oriented computational notebook search tasks. Moreover, we did not explicitly measure the quality of computational notebooks, which may impact the usefulness of the retrieved results. In the future, we will improve the system by introducing computing-related factors into computational notebook ranking, e.g., the executability of computational notebooks/cells and execution time. We will also explore graph-based computational notebook search and recommendation, which leverage the common components within the computational notebooks, e.g., models, datasets, and libraries, to connect computational notebooks. It will potentially provide researchers with high-quality (importance assessment via graph analysis) and personalized (considering the available resources of users) computational notebooks.

Acknowledgment

This work has been partially funded by the European Union's Horizon 2020 research and innovation program by the project CLARIFY under the Marie Skłodowska-Curie grant agreement No 860627, by the ARTICONF project grant agreement No 825134, by the ENVRI-FAIR project grant agreement No 824068, by the BLUECLOUD project grant agreement No 862409, by the LifeWatch ERIC, by the EPSRC grant EP/W032473/1.

References

[1] K. J. O'Hara, D. Blank, and J. Marshall, "Computational notebooks for ai education," 2015.

[2] A. Rule, A. Tabard, and J. D. Hollan, "Exploration and explanation in computational notebooks," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–12.

[3] Z. Zhao, S. Koulouzis, R. Bianchi, S. Farshidi, Z. Shi, R. Xin, Y. Wang, N. Li, Y. Shi, J. Timmermans *et al.*, "Notebook-as-a-vre (naavre): From private notebooks to a collaborative cloud virtual research environment," *Software: Practice and Experience*, vol. 52, no. 9, pp. 1947–1966, 2022.

[4] X. Li, Y. Wang, H. Wang, Y. Wang, and J. Zhao, "Nbsearch: Semantic search and visual exploration of computational notebooks," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–14.

[5] X. Li, Y. Zhang, J. Leung, C. Sun, and J. Zhao, "Edassistant: Supporting exploratory data analysis in computational notebooks with in situ code search and recommendation," *ACM Transactions on Interactive Intelligent Systems*, vol. 13, no. 1, pp. 1–27, 2023.

[6] M. Horiuchi, Y. Sasaki, C. Xiao, and M. Onizuka, "Jupysim: Jupyter notebook similarity search system," *Open Proceedings*, 2022.

[7] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu *et al.*, "Graphcodebert: Pre-training code representations with data flow," *arXiv preprint arXiv:2009.08366*, 2020.

[8] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, M. Gatford *et al.*, "Okapi at trec-3," *Nist Special Publication Sp*, vol. 109, p. 109, 1995.

[9] S. Wang, S. Zhuang, and G. Zuccon, "Bert-based dense retrievers require interpolation with bm25 for effective passage retrieval," in *Proceedings of the 2021 ACM SIGIR international conference on theory of information retrieval*, 2021, pp. 317–324.

[10] J. Guo, Y. Cai, Y. Fan, F. Sun, R. Zhang, and X. Cheng, "Semantic models for the first-stage retrieval: A comprehensive review," *ACM Transactions on Information Systems (TOIS)*, vol. 40, no. 4, pp. 1–42, 2022.

[11] W. X. Zhao, J. Liu, R. Ren, and J.-R. Wen, "Dense text retrieval based on pretrained language models: A survey," *arXiv preprint arXiv:2211.14876*, 2022.

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[13] K. Song, X. Tan, T. Qin, J. Lu, and T.-Y. Liu, "Mpnet: Masked and permuted pre-training for language understanding," *Advances in Neural Information Processing Systems*, vol. 33, pp. 16 857–16 867, 2020.

[14] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih, "Dense passage retrieval for open-domain question answering," *arXiv preprint arXiv:2004.04906*, 2020.

[15] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[16] C. Liu, X. Xia, D. Lo, C. Gao, X. Yang, and J. Grundy, "Opportunities and challenges in code search tools," *ACM Computing Surveys (CSUR)*, vol. 54, no. 9, pp. 1–40, 2021.

[17] F. Lv, H. Zhang, J.-g. Lou, S. Wang, D. Zhang, and J. Zhao, "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)*. IEEE, 2015, pp. 260–270.

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

[19] J. Cambronero, H. Li, S. Kim, K. Sen, and S. Chandra, "When deep learning met code search," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 964–974.

[20] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 933–944.

[21] I. Abdelaziz, J. Dolby, J. P. McCusker, and K. Srinivas, "Graph4code: A machine interpretable knowledge graph for code," *arXiv preprint arXiv:2002.09440*, 2020.

[22] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.

[23] S. Kuzi, M. Zhang, C. Li, M. Bendersky, and M. Najork, "Leveraging semantic and lexical matching to improve the recall of document retrieval systems: A hybrid approach," *arXiv preprint arXiv:2010.01195*, 2020.

[24] L. Gao, Z. Dai, T. Chen, Z. Fan, B. Van Durme, and J. Callan, "Complement lexical retrieval model with semantic residual embeddings," in *Advances in Information Retrieval: 43rd European Conference on IR Research, ECIR 2021, Virtual Event, March 28–April 1, 2021, Proceedings, Part I 43*. Springer, 2021, pp. 146–160.

[25] T. Chen, M. Zhang, J. Lu, M. Bendersky, and M. Najork, "Out-of-domain semantics to the rescue! zero-shot hybrid retrieval models," in *Advances in Information Retrieval: 44th European Conference on IR Research, ECIR 2022, Stavanger, Norway, April 10–14, 2022, Proceedings, Part I*. Springer, 2022, pp. 95–110.

[26] J. Lu, G. H. Abrego, J. Ma, J. Ni, and Y. Yang, "Multi-stage training with improved negative contrast for neural passage retrieval," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021, pp. 6091–6103.

[27] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2019.

[28] M. Pietsch, T. Möller, B. Kostic, J. Risch, M. Pippi, M. Jobanputra, S. Zanzottera, S. Cerza, V. Blagojevic, T. Stadelmann, T. Soni, and S. Lee, "Haystack: the end-to-end NLP framework for pragmatic builders," Nov. 2019. [Online]. Available: https://github.com/deepset-ai/haystack

[29] Z. Zhao, S. Koulouzis, R. Bianchi, S. Farshidi, Z. Shi, R. Xin, Y. Wang, N. Li, Y. Shi, J. Timmermans *et al.*, "Notebook-as-a-vre (naavre): from private notebooks to a collaborative cloud virtual research environment," *arXiv preprint arXiv:2111.12785*, 2021.

[30] N. Li, Y. Zhang, and Z. Zhao, "Cnsvre: A query reformulated search system with explainable summarization for virtual research environment," in *Companion Proceedings of the ACM Web Conference 2023*, 2023, pp. 254–257.

[31] P. Bajaj, D. Campos, N. Craswell, L. Deng, J. Gao, X. Liu, R. Majumder, A. McNamara, B. Mitra, T. Nguyen *et al.*, "Ms marco: A human generated machine reading comprehension dataset," *arXiv preprint arXiv:1611.09268*, 2016.

[32] J. M. Perkel, "Why jupyter is data scientists' computational notebook of choice," *Nature*, vol. 563, no. 7732, pp. 145–147, 2018.