Software: Practice and Experience

WILEY

RESEARCH ARTICLE OPEN ACCESS

Search Multiple Types of Research Assets From Jupyter Notebook

Na Li[1] | Siamak Farshidi[1,2] | Zhiming Zhao[1,3]

[1]Multiscale Networked System, University of Amsterdam, Amsterdam, Netherlands | [2]Information Technology Group, Wageningen University and Research (WUR), Wageningen, Netherlands | [3]LifeWatch ERIC, Virtual Lab and Innovation Center (VLIC), Amsterdam, Netherlands

Correspondence: Zhiming Zhao (z.zhao@uva.nl)

Received: 27 August 2024 | **Revised:** 29 November 2024 | **Accepted:** 17 December 2024

Funding: This work was supported by the EU H2020 CLARIFY project (860627), EU LifeWatch ERIC, Dutch Research Council, LTER-LIFE project, EU Horizon Europe BLUECLOUD 2026 project (101094227), and EU Horizon-Europe EVERSE project (101129744), EU Horizon-Europe ENVRI-HUB next project (101131141).

Keywords: computational notebook search | data discovery | data science | dataset search | Jupyter notebook | Multiple Research Asset Search

ABSTRACT

Objective: Data science and machine learning methodologies are essential to address complex scientific challenges across various domains. These advancements generate numerous research assets such as datasets, software tools, and workflows, which are shared within the open science community. Concurrently, computational notebook environments like Jupyter Notebook, along with platforms like Google Colab and Kaggle Kernel, facilitate data science research and machine learning workflows, transforming data analysis, model development, and knowledge sharing processes. The proliferation of computational notebooks has further enriched the pool of valuable research assets. Researchers frequently require efficient access to these assets to advance their work, yet current tools often require navigating multiple websites and portals, leading to inefficiency and information overload. The challenge is compounded when relying on general web search engines that might not adequately highlight niche scientific resources.

Methods: To address these issues, we propose the development of an innovative Multiple Research Asset Search (MRAS) system designed to index diverse research assets from heterogeneous sources, offering a unified search interface for researchers. Our system aims to significantly improve the discovery of computational notebooks and datasets, facilitating data-driven research.

Results: We developed a pipeline for data extraction and indexing, reviewed and applied state-of-the-art ranking algorithms, enhanced indexing documents with content analysis, and created a Jupyter extension for asset discovery within the working environment.

Conclusion: This work is structured to detail our approach, literature review, system development, empirical validation, results, and conclusions, illustrating the potential impact of our MRAS system on scientific research efficiency.

1 | Introduction

Data science and machine learning (ML) approaches play a pivot role in studying complex scientific problems, for example, identifying cancer from pathological images [1], modeling the evolution of ecosystems [2], and developing scheduling policies for dynamic computing tasks [3]. As these fields advance, a substantial amount of research artifacts—including

Abbreviations: AI, artificial intelligence; API, application interface; CERN, European Organization for Nuclear Research; JSON, JavaScript Object Notation; ML, machine learning; MRAS, Multiple Research Asset Search; MVT, Model-View-Template; NaaVRE, Notebook-as-a-VRE; SERP, Search Engine Results Page.

datasets, software tools, and workflows—are created and shared within the open science community, making them valuable *research assets*.

Meanwhile, computational notebook programming environments like Jupyter Notebook provide powerful frameworks that support data science research and ML workflows. Platforms such as Google Colab[1] and Kaggle Kernel[2] offer hosted Jupyter Notebook services, granting free access to computing resources, while NaaVRE [4] offers a collaborative virtual research environment based on Jupyter Notebook. These notebook-based solutions have transformed how researchers analyze data, develop models, and share knowledge. With widespread adoption, numerous computational notebooks have been generated and published, making them another essential category of research asset.

Scientists often need to locate these assets to support their research activities, demanding efficient tools and methods for discovery and utilization. For instance, a researcher working on developing a new ML model for medical image analysis might require access to a specific dataset of annotated pathological images. Finding the right dataset and computational notebooks for data analysis can significantly expedite their research, allowing them to design, train, and validate their models effectively. However, with the current tools, researchers usually have to navigate multiple websites and portals to collect necessary research assets scattered across different data sources, which can be time-consuming and ineffective. Besides, the sheer volume of available research assets can lead to information overload, making it challenging for them to efficiently identify relevant resources. The situation worsens when researchers have to resort to general-purpose web search engines but the desired resources sit in the "long tail" of web content [5].

To address these challenges, we aim to develop an innovative Multiple Research Asset Search (MRAS) system capable of indexing various types of research assets from diverse data sources, enabling users to discover a wide range of research resources through a single search interface. Our system primarily targets the science domain with a prominent goal of building paradigm-shifting information software for scientific researchers. Although the proposed system can handle various types of research assets, this study specifically focuses on *computational notebooks* and *datasets*. Computational notebooks align seamlessly with a Jupyter-based research environment, and datasets are indispensable for data-driven studies.

Unlike many existing studies that primarily emphasize code search using various enhancement techniques such as query expansion [6], software repository mining [7], and code summarization [8], our work focuses on delivering a holistic research asset discovery solution. This approach not only includes source code but also encompasses datasets and other valuable assets. Additionally, while some previous works address research asset management [4] and reproducibility [9], their emphasis is more on computational processes rather than on improving asset discoverability. To guide our research, we identify several key research questions as listed below:

1. How can we extract and index diverse research assets across multiple sources?
2. How to effectively rank research assets such that the most relevant/useful assets will be presented on the top?
3. How can computational notebooks be effectively represented in retrieval systems, considering their task-oriented nature and the combination of natural language and programming language elements?
4. How to integrate this system effectively into the Jupyter environment to support seamless research workflows?

For the first question, we developed a data extraction and indexing pipeline for each type of research asset to collect data from various data providers and established a central knowledge base to integrate different asset indexes. For the second question, we reviewed related studies on ranking algorithms and exploited state-of-the-art ranking algorithms for research asset retrieval. For the third question, we enhanced the indexing documents with important information extracted from the content of computational notebooks through content analysis. For the fourth question, we developed a Jupyter extension with a search interface in the Jupyter environment that can communicate with our MRAS system through RESTful API calls. It supports the discovery of outside research assets inside researchers' working space without platform switching.

This article is organized as follows. Section 2 presents the literature study, in which we review the start-of-art work and related work. Section 3 describes the proposed MRAS system to bridge the gap between researchers' needs for research asset discovery and the inefficiency of current search tools. In Section 4, we introduce the empirical evidence to demonstrate the applicability of our system in various domains and usage scenarios. Section 5 presents the results and analysis of our system. Finally, in Sections 6 and 7, we summarize this work with discussions, conclusions, and future work.

2 | Literature Study

Research asset discovery is a broad and interdisciplinary field that requires a variety of knowledge and methodologies. Before embarking on designing and developing a search system aimed at aiding scientists in finding relevant research assets, it is essential to first comprehend existing solutions and pinpoint discrepancies between these solutions and our objectives. Another critical step involves gaining a thorough understanding of prior work relevant to constructing a research asset search system, empowering us to leverage optimal algorithms and technologies. To accomplish these objectives, a comprehensive literature review is conducted. This section will outline state-of-the-art methods and tools for research asset discovery, highlight their limitations, and examine relevant studies on search techniques.

2.1 | State of the Art

This section will explore how researchers typically find existing research materials. We will start by looking at common strategies and the tools researchers exploit to locate relevant resources. Then, we will focus on state-of-the-art methods applied specifically for finding computational notebooks and datasets, which

are the main type of research assets we are interested in for this study.

2.1.1 | Research Asset Search Practices

Currently, the search for research assets is mainly enabled by three types of tools. The first type includes general-purpose web search engines like Google Search[3] and Microsoft Bing,[4] which serve everyday users but lack specialization in scientific domains. The second type consists of domain-specific catalogs, such as ICOS[5] for greenhouse gas datasets and LifeWatch[6] catalogs for software services. The third type is asset repositories equipped with search functionalities. For instance, Kaggle supports search over computational notebooks and datasets that are hosted on Kaggle. Zenodo[7] allows search over publications, software, datasets, images, presentations, and more that are published through Zenodo. While web search engines are useful for broad initial searches, they often fall short in locating the precise resources needed for specific research tasks. On the other hand, domain-specific catalogs excel in offering precise resource discovery but are restricted to specific asset types. Although repositories like Zenodo and Kaggle support multiple research assets, they only allow platform-wise search and cannot access resources from other platforms. Research activities typically require multiple types of research assets that are hosted by different platforms, and hence researchers often spend a significant amount of time traversing different search engines, catalogs, and repositories to collect sufficient resources. Therefore, it is essential to build a well-integrated search system supporting searches across various categories of research assets from different data sources.

2.1.2 | Computational Notebook Search

Regarding computational notebooks, web search engines remain widely used. Computational notebooks are typically regarded as open-source codes and thus code repositories such as GitHub and Kaggle are also instrumental in discovering computational notebooks. However, these tools mainly facilitate indirect searches for computational notebooks (e.g., web search engines) or within specific data sources (e.g., code repositories). Recently, dedicated studies for searching computational notebooks have emerged, such as DeCNR [10] and CNSVRE [11]. DeCNR improves computational notebook ranking accuracy by combining Best Matching 25 (BM25) [12] and SBERT [13] methods. CNSVRE enhances overall search efficiency through query reformulation and notebook summarization. Other partially relevant tools involve Nbsearch [14], JupySim [15], and EDAssistant [16]. Nbsearch and EDAssistant focus on code snippet search within notebook collections, categorizing them as code search rather than notebook search tools. JupySim, on the other hand, is a content-based search system that retrieves notebooks based on specified contents like codes, data, libraries, and output formats, distinct from the natural language-based notebook search problem addressed by CNSVRE and DeCNR.

2.1.3 | Dataset Search

There are two types of dataset search tools: vertical dataset search engines and dataset repositories. Vertical dataset search engines usually do not store datasets but rely on harvested dataset metadata to rank datasets. Examples are Google Dataset Search [5] and Auctus [17]. Dataset repositories often support storing, sharing, and searching datasets. Examples are Zonodo, Data.gov, and Kaggle. Google Dataset Search [5] provides web-wide search capabilities to discover publicly available datasets across different Web locations, from governmental and community dataset repositories to individual data providers. It employs Schema.org and W3C DCAT markup to find datasets published on the Web. Auctus [17] is an open-source dataset search engine for structured data. It supports various queries, including keywords, spatial and temporal information, and data integration queries. Unlike Google Dataset Search, it does not collect datasets through web documents but hooks up with dataset repositories, such as Socrata and Zenodo, via APIs. Zenodo is a digital repository and open-access platform designed to facilitate the sharing, preservation, and dissemination of scientific research outputs and data. It was developed by European Organization for Nuclear Research (CERN) and is now operated by the OpenAIRE project. Data.gov is a comprehensive platform maintained by the U.S. government that aggregates a vast collection of datasets from various government agencies. It provides a user-friendly interface for searching and accessing open data on topics ranging from healthcare to environment and more. Kaggle dataset offers a repository of datasets contributed by the data science community.

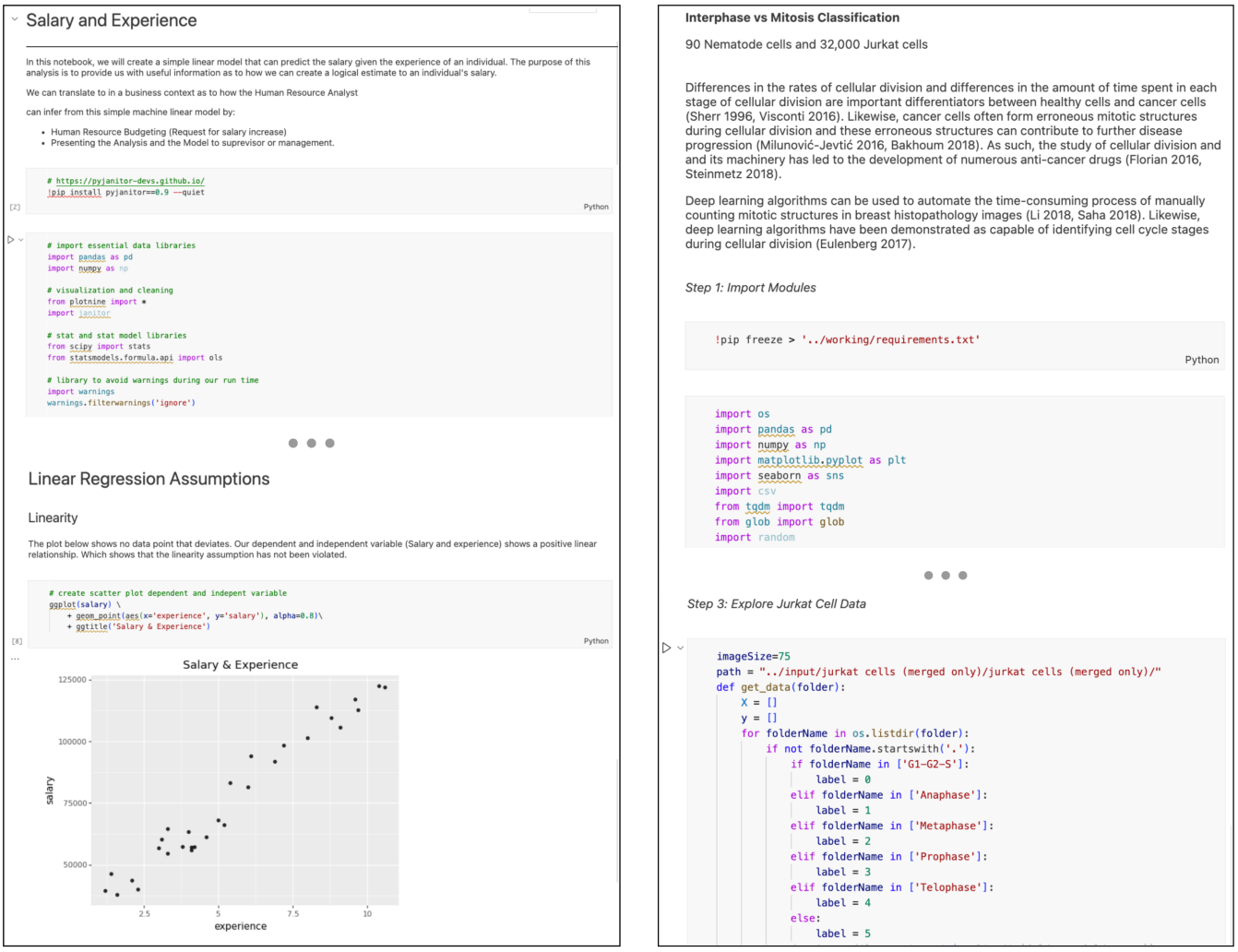
In summary, researchers often rely on web search engines to search for existing research assets. Despite the widespread use of Jupyter Notebooks, a dedicated computational notebook search software is lacking. While there are public vertical search engines for dataset search, these are not integrated with other search functions for different types of research assets, making them inefficient in building data analytic workflows.

2.2 | Related Work

We have identified deficiencies in the current search tools available for various research resources, particularly for computational notebooks and datasets. Our objective is to create a unified search platform that includes all these assets and can be integrated with a Jupyter Notebook environment to support research activities. A critical component of any search system is its result ranking mechanism. To enhance our system's performance, it is crucial to understand the different ranking techniques used for each category. Therefore, we have studied search methodologies specific to computational notebooks, datasets, and those applicable across multiple categories of research assets.

2.2.1 | Computational Notebook Search Methods

Computational notebooks essentially represent multi-modal data that primarily include free text and source code, with optional elements such as images and audio. They organize content into segments called *cells*. There are two fundamental types of cells: *code cells*, which contain code fragments, and *Markdown cells*, which provide narrative descriptions. We present two examples of computational notebooks in Figure 1 to illustrate



(a) One example

(b) Another example

FIGURE 1 | Computational notebook examples. (a) Simple linear regression model and assumptions. *Source:* https://www.kaggle.com/code/jaepin/simple-linear-regression-model-assumptions. (b) Interphase vs. mitosis classification. *Source:* https://www.kaggle.com/code/paultimothymooNey/interphase-vs-mitosis-classification.

their typical content structure. Computational notebooks can be searched using metadata, the code within code cells, or the textual descriptions in Markdown cells. This section will briefly review the existing methods and technologies for searching computational notebooks.

2.2.1.1 | Searching Computational Notebooks Via Metadata. The necessity of providing metadata for computational notebooks varies based on the publication platform. Authors might need to include contextual information through metadata when publishing on certain platforms. For instance, Kaggle attaches metadata to each computational notebook, while GitHub typically includes computational notebooks as part of larger software projects and provides metadata for the entire project. These metadata can be leveraged to build a computational notebook discovery system. High-quality metadata enables users to conduct facet-based searches and more effectively find relevant results. However, metadata does not convey as much information as the actual content, such as the text and code within the notebooks.

2.2.1.2 | Searching Computational Notebooks Via Markdown Cells. Markdown cells contain natural-language descriptions of the experiment purpose, logic, or code functionality in the computational notebooks. Searching computational notebooks via Markdown cells can be seen as a text retrieval [18] problem, which aims to find the most relevant documents concerning given queries. Relevancy is measured by the similarity between documents and queries. There are two main types of methods for text retrieval: *sparse retrieval* and *dense retrieval*, distinguished by the vector representations of text [18]. Sparse retrieval methods use a sparse vector, for example, one-hot encoding (all vector elements are 0 s except that one element is 1, and vector dimension depends on the vocabulary size), to represent a word. Term Frequency-Inverse Document Frequency (TF-IDF) [19] and BM25 are exemplar sparse retrieval methods and are usually used as baseline methods for text retrieval. More specifically, BM25 ranks a set of documents based on the statistical information of query terms appearing in the documents, disregarding grammar and word order of queries and documents (bag-of-words assumption). In contrast, dense retrieval methods [20] represent

word/sentences/documents with learned dense low-dimensional vectors, for example, DPR [21] and ColBERT [22]. Sparse retrieval performs well on precise term matching, and dense retrieval is better at measuring semantic similarities between related words.

2.2.1.3 | Searching Computational Notebooks Via Code Cells. Searching computational notebooks based on code cells is often formulated as a *code search* problem [23] which retrieves code fragments from a large code corpus that most closely matches users' intents that are expressed in natural language queries. The code search problem is mainly studied in the software engineering field to improve software developers' productivity by facilitating the reuse of existing codes. Existing code search methods can be roughly classified into two groups: traditional Information Retrieval (IR) based [24] and ML based [25–27]. Traditional IR-based code search methods usually treat code as text and use statistical ranking algorithms such as TF-IDF [19] and BM25 [12] to compute the similarity scores between queries and code snippets. ML-based methods map both codes and queries to real-valued vectors and measure the relevancy using cosine similarity between vector representations [28]. Traditional IR-based methods are easy to implement but highly rely on shared vocabulary between queries and codes. On the other hand, ML-based methods can reduce the semantic gap between queries and codes but generally require a time-consuming model training process.

2.2.1.4 | Summary. Searching computational notebooks via metadata is the simplest way and is commonly adopted by computational notebook catalogs and repositories. Metadata provides contextual information, for example, provenance of computational notebooks but usually lacks details about the contents. Searching via Markdown cells simplifies the problem into a text retrieval problem which makes it possible to apply state-of-the-art text retrieval algorithms. However, neglecting code contents may cause high rankings of low-reusable computational notebooks, for example, computational notebooks with rich text descriptions but low-quality codes or no codes. Searching via code cells, on the other hand, may cause ineffective ranking as text descriptions are indispensable for users to find and comprehend the computational notebooks, even though searching computational notebooks often leads to the reuse of the codes. In summary, we solve the computational notebook search problem by analyzing, indexing, and ranking the contents of the computational notebooks including texts in markdown cells and codes in code cells. Meanwhile, metadata remains and is used to present search results.

2.2.2 | Dataset Search Methods

As pointed out by [29], "dataset search is largely keyword-based over published metadata." Given the variety of dataset contents, utilizing structured metadata that contains textual descriptions facilitates more efficient indexing and ranking. In this study, we adopt the common practice in the dataset search domain by focusing our search on metadata. Our innovation lies in the integration and consolidation of metadata from various heterogeneous data sources.

3 | Multiple Research Asset Search System

3.1 | Requirements Elicitation

Developing an effective MRAS system requires the understanding of scientists' inquiries in order to help them find desirable research resources. Consider Elena, an Artificial Intelligence (AI) researcher specializing in histopathological image analysis, particularly focusing on mitosis segmentation. She intends to explore the application of Conditional Generative Adversarial Network (cGAN) in the mitosis detection task, starting with building a data analysis workflow via Jupyter Notebook. This workflow involves using cGAN to recognize mitotic cells within H&E stained histopathological images. To accomplish this task, she must gather all the necessary resources, including histopathological images, computational notebooks/codes for cGAN implementation, and descriptions of the data processing procedure. In reality, one or more of these materials are often missing or insufficient, necessitating the search for external research assets. Elena's case serves as a prime example of how our MRAS system can be utilized. To better assist researchers like Elena, we have outlined key requirements for developing an effective MRAS system.

3.1.1 | The Source of the Requirements

The requirements are primarily derived from literature study, interviews, and user surveys. The extensive review of existing literature helps identify gaps between current search systems and the goal of the MRAS system. Interviews are conducted with scientists and stakeholders from the EU CLARIFY project collaborators to gain insights into practical challenges faced by researchers and the features they consider most critical for an effective search system. CLARIFY, an EU Horizon 2020 initiative, aims to create a robust automated digital diagnostic environment using AI and cloud-based data algorithms to improve Whole Slide Image (WSI) interpretation and diagnosis. The project includes partners from three key backgrounds: AI scientists, data scientists, and pathologists. The interviews took place through regular meetings, informal gatherings, and other interactions during one of the author's 3-month visits to two collaborating research groups. Specifically, discussions were held with researchers from various fields, including medical image analysis and IR to learn about their routine research workflows and explore how a MRAS system could support their research activities. We also survey needs, preferences, comments, and suggestions from internal users, mainly consisting of researchers and software developers within our research team. During the system development process, these users actively participated in regular team discussions to review progress and provide valuable feedback and input for the system's design and implementation. After analyzing and consolidating the input from the participants, we identify four functional requirements and two nonfunctional requirements. Functional requirements describe the specific behaviors, functions, and operations that a system must perform, whereas nonfunctional requirements describe the qualities, attributes, and characteristics that the system must have.

3.1.2 | Functional Requirements

Functional requirements for the MRAS system include:

A central knowledge base with a unified search interface. A prominent feature appreciated by researchers is having a single entry point to access a vast volume of research assets. To achieve this, a central knowledge base capable of integrating various types of research assets from multiple sources, along with a unified search interface, is essential. The knowledge base should be able to process assets in different formats, metadata schemas, and access protocols. Meanwhile, the search interface can simplify the discovery of relevant resources, thereby enhancing researchers' productivity.

Effective ranking for research assets. Effective ranking algorithms can alleviate the information overload resulting from the enormous number of computational notebooks, datasets, and other research materials in the search space. These algorithms provide researchers with a ranked list of resources tailored to their specific needs and interests. To achieve this, natural language processing algorithms like word embedding and IR techniques like dense retrieval can be leveraged.

Content analysis for computational notebooks. Researchers emphasize the importance of sufficient descriptions of computational notebooks to assess their relevance and usability. The descriptions should address aspects such as targeted tasks, utilized algorithms, and other statistical details, allowing users to quickly understand its purpose, methodology, and key findings. This level of information can hardly be obtained from mere metadata, but can be extracted from computational notebooks' contents. So a content analysis process is necessary for the MRAS system to facilitate the ranking efficiency and better result presentation for computational notebooks.

Integration with Jupyter Notebook. Researchers tend to stick with familiar tools for their work. For many, Jupyter Notebook is their primary environment for tasks like data exploration, preprocessing, model development, and evaluation. Providing search functionality within the Jupyter Notebook environment would seamlessly embed search operations into their research routines. Therefore, we aim to integrate the MRAS system with Jupyter, enabling scientists to transition smoothly between the search interface and their data pipeline.

3.1.3 | Nonfunctional Requirements

Nonfunctional requirements for the system include:

Comprehensive metadata for result display. In practice, the search result page does not display the full content of the returned research assets due to space constraints. Instead, it presents key information, such as the title, a brief summary, and attributes like author information, asset size, and license. This makes it challenging for researchers to assess the quality and usability of the assets. To address this issue, the MRAS system should enhance its metadata with the necessary information for better result presentations.

Continual improvement with user feedback. Continuous improvement is one important characteristic of any user-interactive system, and user feedback is a crucial component for achieving it. To facilitate this, the system should provide efficient tools for collecting user feedback on search results, allowing users to influence the ranking algorithms. Additionally, the system should be capable of analyzing user interactions and feedback to continually enhance its algorithms, creating a self-improving mechanism that refines the ranking and presentation of research assets over time.

3.2 | Conceptual Framework

This section outlines the conceptual framework for a MRAS system, introducing its main functional components and high-level workflow, while avoiding excessive details such as software design, infrastructure specifics, and ranking algorithms. The current focus of research assets includes computational notebooks and datasets, and therefore, we will describe each component in relation to these two types of assets. However, our approach is adaptable to other types of research assets such as scientific articles, multimedia materials, and web APIs.

Figure 2 depicts the proposed framework. The system is divided into several key parts: The *data processing and indexing* pipeline gathers, preprocesses, and indexes data from diverse sources. The *data integration* module consolidates indexes of various research assets and user data. The *asset retrieval* unit handles user queries and ranks research assets based on the queries. The *user interface* collects user input and displays the retrieved research assets.

3.2.1 | Data Processing and Indexing

The *data extraction* operation gathers computational notebooks along with their metadata and dataset metadata from different online sources using a web crawler that leverages platforms' APIs. Major platforms hosting computational notebooks include GitHub and Kaggle, while datasets are commonly found on platforms like Kaggle, Zenodo, and Figshare. Computational notebooks and datasets obtained from various sources often adhere to different metadata standards. To consolidate these resources, *preprocessing* is necessary. This step transforms the raw records downloaded from hosting repositories into a standardized format for each type of asset. Additionally, the preprocessing stage can extract additional information from the contents of computational notebooks to enrich the indexes. The *indexing* process then creates asset indexes based on the processed documents generated by the preprocessor, enabling efficient retrieval of assets.

3.2.2 | Data Integration

We implement tailored data extraction and indexing pipelines for each individual data source. These pipelines are designed to handle the specific characteristics and formats of data from different sources. Upon completion of these pipelines, multiple indexes are generated, each corresponding to the data retrieved from its respective source. Subsequently, we utilize a *knowledge base* to manage these indexes along with user data gathered from interactions between users and the system. The knowledge base serves as a central repository where these indexes are stored and managed, allowing for efficient organization and retrieval of information across various sources.

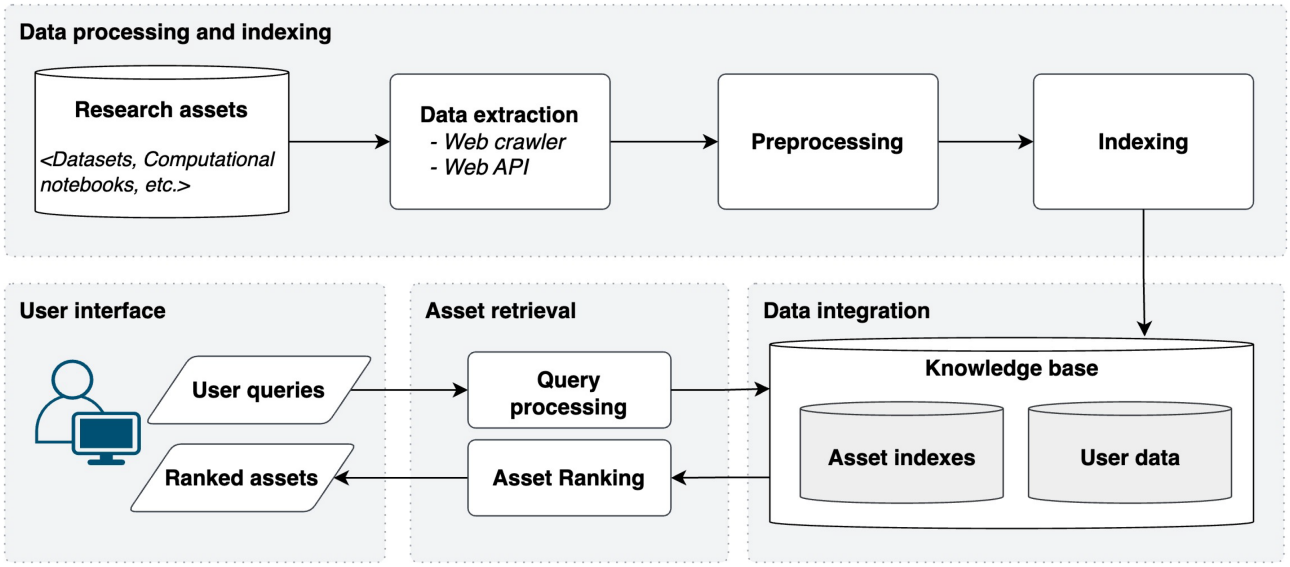


FIGURE 2 | The conceptual framework of the Multiple Research Asset Search system.

3.2.3 | Asset Retrieval

The *asset retrieval* module processes user queries and returns ranked lists of research assets, with items ordered by their similarity scores (higher scores result in higher rankings). The ranking is determined by comparing the queries to the content in selected fields. Choosing retrieval methods involves balancing ranking accuracy with system responsiveness. If the ranking is inaccurate, irrelevant results may appear at the top, reducing users' confidence in the system. On the other hand, complex ranking algorithms can cause expensive computation costs and high system latency, diminishing usability. Therefore, system designers must carefully weigh the trade-offs between accuracy and responsiveness, considering the specific requirements and resource constraints.

3.2.4 | User Interface

The *user interface* serves as the entry point for users to interact with our system. We offer two types of interfaces. One is the web-browser-based interface that allows manual manipulations of the research assets, such as browsing and selecting. It features a search bar for entering queries and a Search Engine Results Page (SERP) for displaying the retrieval results. Another one is web Application Interface (API) that enables programmable access to the system. Unlike the browser-based interface, it does not include web pages for result presentation. Instead, it focuses on the comprehensiveness and programmability of the results, making it suitable as a building block for other tools and web services.

3.3 | Technical Considerations

Besides the requirements for meeting user's needs and expectations, implementing a MRAS system also involves various technical considerations to ensure its effectiveness, scalability, and usability. Here are key technical considerations:

Ranking algorithm complexity vs. system responsiveness. We have observed a common trade-off between implementing sophisticated ranking algorithms and ensuring fast system responses given computing resource constraints. Current advanced retrieval methods for texts and codes heavily rely on dense document representations, which involve transforming documents into fixed-length high-dimensional vectors, also known as embeddings, and measuring their similarities based on these vector distances [21]. These methods impact system performance in two main ways. Firstly, the models used for generating embeddings are typically complex and computationally intensive. While conducting the embedding generation process offline can reduce the workload on the online system, these procedures remain resource-intensive and challenging to maintain. Secondly, performing searches within a vector space often requires in-memory computations to enhance search efficiency, posing scalability challenges. Therefore, system designers must carefully balance algorithmic complexity and system responsiveness to ensure a satisfactory user experience.

Metadata standardization vs. metadata informativeness. As a MRAS system facilitates searches spanning heterogeneous sources, the metadata associated with accumulated research assets may conform to different schemas. Without standardization, the system would need distinct indexes for each source, leading to significant hindrances in database consistency and management. However, developing a universal metadata format that can accommodate diverse schemas and provide sufficient information for asset indexing presents a considerable challenge. To tackle this problem, we suggest implementing a foundational metadata schema for each research asset specifically for indexing and retrieval purposes, while preserving the original metadata for users to access directly.

4 | Prototype

In our latest prototype, we have created and activated search functionalities tailored for computational notebooks and datasets,

fulfilling critical needs across multiple disciplines. The system architecture, currently implemented for these two types of research assets, can be easily extended to include other types of research assets. Moreover, we envision our system evolving into an online data platform to enhance research asset discovery. We have developed a data collection mechanism accompanying the main functionalities of the MRAS system. Specifically, we designed several data models to organize essential information, created input components on the user interface, and deployed a user database to manage gathered data efficiently.

4.1 | Functionality Implementation

We obtain computational notebooks from GitHub and Kaggle using their public APIs. The source files of computational notebooks are JavaScript Object Notation (JSON) objects that follow a common schema, allowing us to efficiently digest computational notebooks from different repositories. We gather dataset metadata records from Kaggle, Zenodo, and Dryad. However, the metadata schemas from these platforms vary significantly. Figure 3 shows the subset of metadata from two random datasets hosted on Kaggle and Zenodo, which have few common field names even for those with similar contents. For instance, the identifier of the dataset is named "ref" on Kaggle and "conceptdoi" on Zenodo. Kaggle denotes creator name simply as "creatorName" while Zenodo uses a nested field "metadata.creators" to represent all the creator names. To address this discrepancy in metadata schemas, we develop a customized metadata parsing protocol for each data source.

We follow the data preprocessing procedure depicted in Figure 4 to enhance the indexing content of computational notebooks and to standardize the metadata of datasets obtained from various


```
"ref": "harmandeepsinghpadam/clips-with-
annotation",
"id": 3710582,
"title": "Count the Clips (Object detection) ",
"hasTitle": true,
"subtitle": "Clips images and yolo labels",
"hasSubtitle": true,
"creatorName": "Harmandeep Singh Padam",
"hasCreatorName": true,
"creatorUrl": "harmandeepsinghpadam",
"hasCreatorUrl": true,
"totalBytes": 400864,
"hasTotalBytes": true,
"licenseName": "CC BY-SA 4.0",
"hasLicenseName": true,
"description": "",
"hasDescription": false,
"downloadCount": 0,
"viewCount": 31,
"voteCount": 1,
```


(a)


```
"conceptdoi": "10.5281/zenodo.818933",
"conceptrecid": "818933",
"metadata": {
    "creators": [
        {
            "name": "PowerTAC"
        }
    ],
    "description": "Log and boot files of game 50",
    "keywords": [
        "PowerTAC"
    ],
    "license": {
        "id": "CC-BY-4.0"
    },
"stats": {
    "downloads": 10.0
    "views": 21.0,
    "volume": 1546298850.0
},
```


(b)

FIGURE 3 | Different metadata schemas utilized by different dataset hosting platforms. (a) Dataset metadata example from Kaggle. *Source:* https://www.kaggle.com/datasets/harmaNdeepsinghpadam/clips-with-annotation. (b) Dataset metadata example from Zenodo. *Source:* https://zenodo.org/records/818934.

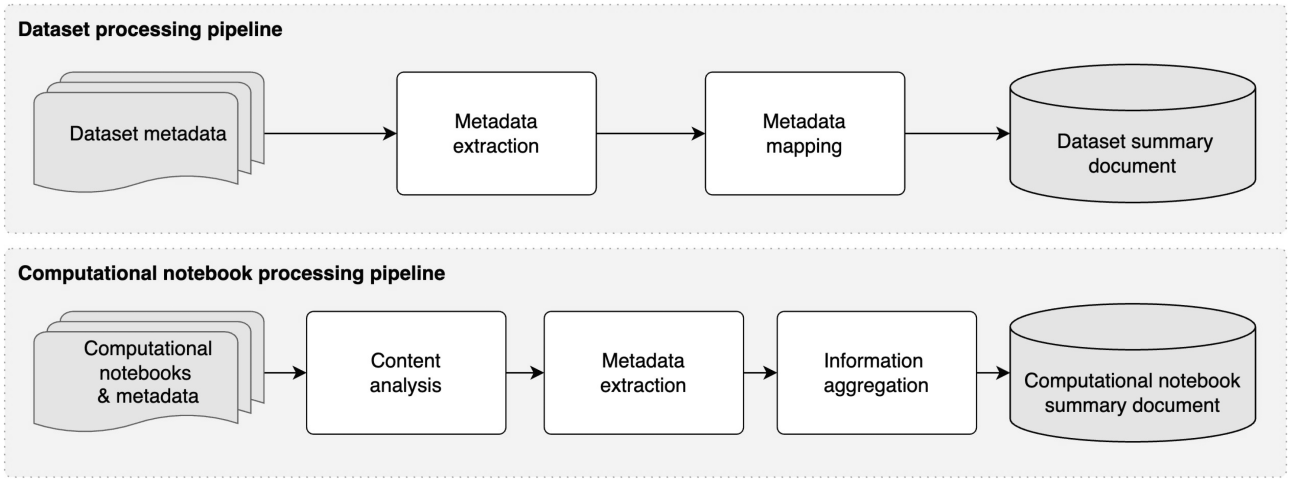


FIGURE 4 | Preprocessing pipelines for datasets (top), and for computational notebooks (bottom).

hosting repositories. For computational notebooks and their associated metadata files, the preprocessor first analyses the raw contents of computational notebooks by parsing the source files to extract useful information. For instance, it extracts the textual contents from all Markdown cells as the "description" of the computational notebook and utilizes the T5 model [30] for summarization [11]. It also produces several statistical metrics, such as the number of code cells, the number of Markdown cells, and the programming language. Then it gathers administrative information such as "name," "full_name," and "html_url" from the corresponding metadata files. Finally, it aggregates all the information generated within the above procedures into one single structured document called *computational notebook summary document*. For dataset preprocessing, a key operation is *metadata mapping*, which is essential for consolidating datasets from heterogeneous sources [31]. The processor first extracts a subset of fields and values from the dataset metadata. These field-value pairs are then mapped to a common dataset metadata schema yielding a *dataset summary document* to facilitate indexing and retrieval.

Considering the system's computing overhead, we employ inverted indexes for indexing and the BM25 algorithm for ranking. Inverted indexes facilitate efficient search queries by mapping terms to their respective documents, optimizing both search speed and storage. BM25 is a simple yet highly effective ranking method, which considers term frequency, document frequency, and length normalization to ensure highly relevant search results. The combination of inverted indexes and BM25 allows for efficient retrieval of research assets in a vast search space. We leverage Elasticsearch during deployment to achieve the production system level. It integrates these technologies into a scalable and responsive solution that can handle large datasets and high query loads.

The backend of the MRAS system is developed as a Django web application. It handles retrieval for various types of research assets, generates SERPs for web browsers, and offers RESTful API services. The application connects with Nginx, Elasticsearch, and PostgreSQL for necessary data exchange. Django enforces a clear separation of concerns between different components of a web application, often referred to as the Model-View-Template (MVT) architectural pattern. In this pattern, templates handle the presentation layer, views manage the application logic and data processing, and models deal with the database and data storage. To facilitate data exchange with relational databases and API calls, we use models and serializers to define data structures and generate structured data. Models represent the structure and behavior of database tables, defining fields, relationships, and methods for working with data. Serializers convert complex data types, such as querysets and model instances, into JSON or other content types. For example, we define a *NotebookResult* model to represent the search results of computational notebooks, comprising context information and a list of retrieved records. This model works together with the serializer *NotebookResultSerializer* to serialize the data for inclusion in an API response.

4.2 | Content Management

Effective data management is key to minimizing system latency and enhancing the usability of a MRAS system. This section describes the content management procedure within our system, as illustrated in Figure 5. The process incorporates three storage options: disk storage, index database, and user database. With a primary focus on the online search system rather than offline processes, data management during crawling and preprocessing is simplified. The crawler stores downloaded source files of research assets along with their metadata files in disk storage.

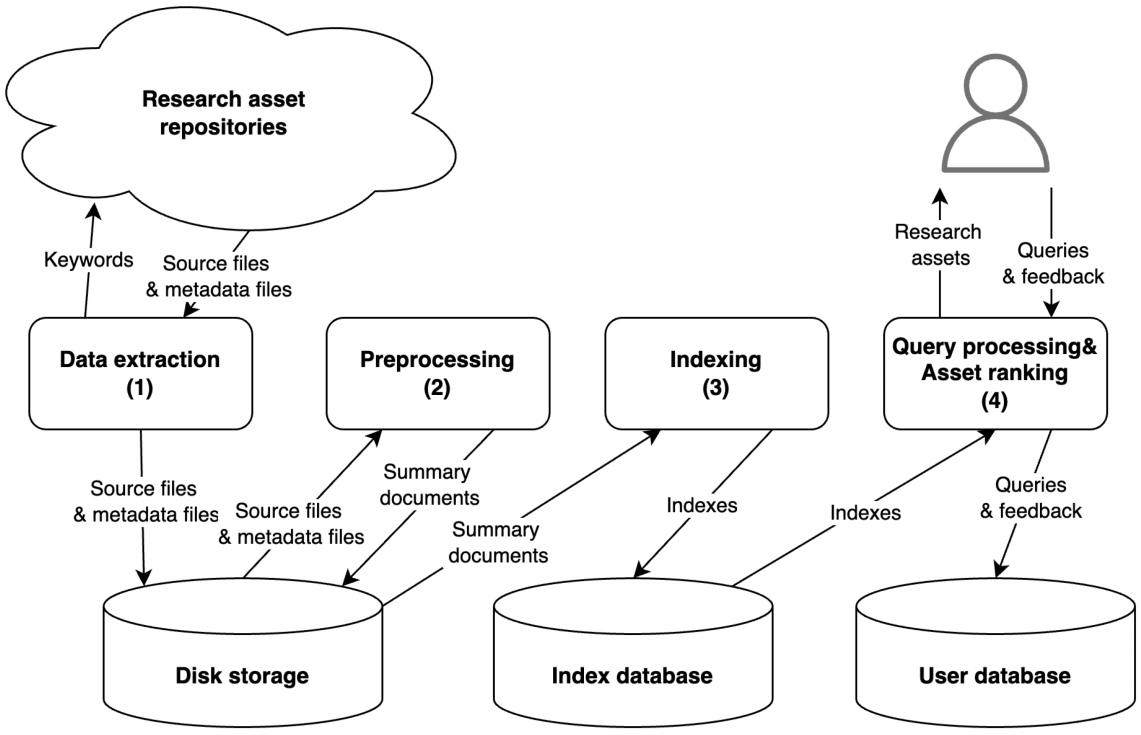


FIGURE 5 | Content management procedure.

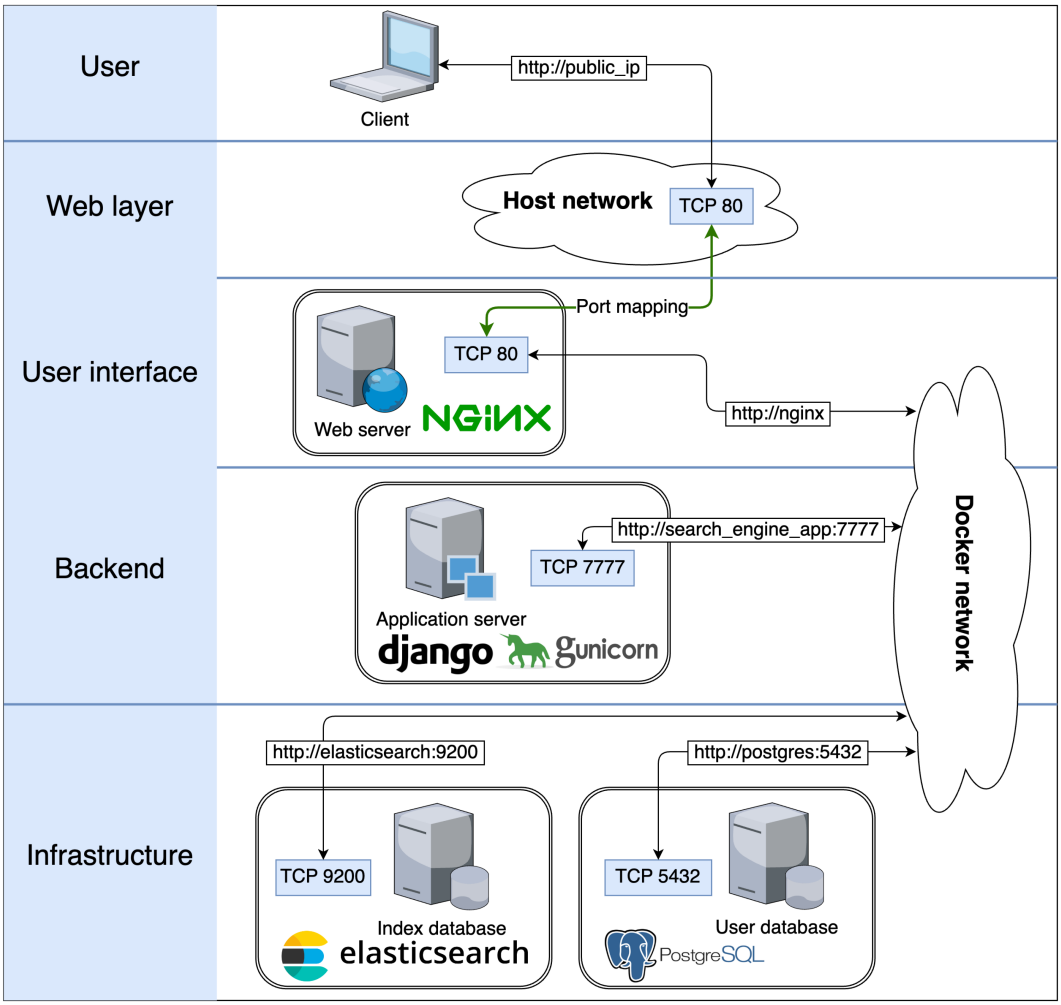


FIGURE 6 | Infrastructure and network setups for system deployment.

Subsequently, the preprocessor reads raw data from the disk, processes it, and writes summarized documents of research assets back to disk storage. This approach is optimized for high-volume, low-frequency tasks, such as batch processing of research assets. The indexing pipeline generates indexes from the summary documents of each research asset and stores them in the index database. The index database's main role is to facilitate fast document retrieval by structuring and organizing data in an index format, allowing quick access to specific records without full-table scans. Alongside the asset ranking module, which fetches relevant records from the indexes, the query processing unit logs users' queries and feedback into the user database when necessary. We use Elasticsearch as the index database due to the need to process free-form data and handle large data volumes in near real-time. The user database manages structured data rather than unstructured text and primarily functions as storage with less emphasis on frequent searches, for which we employ PostgreSQL.

4.3 | System Deployment

We present the infrastructure architecture utilized by our current prototype in the deployment environment, as depicted in Figure 6. The infrastructure is divided into several layers to clearly define the responsibilities of each module. Each module is implemented either as a server or a database. A web server manages web traffic, an application server implements the core system functionalities described in Section 3.2, an index database handles various indexes, and a user database stores interactive data.

To enhance the portability and scalability of our system, all servers and databases are run within Docker containers. Docker offers a containerization platform for application deployment, enabling developers to package an application and its dependencies into a container for consistent, isolated, and portable deployment across diverse environments. To manage communication between servers and between servers and databases, an efficient mechanism is required to allow them to share data and collaborate on different tasks. We use two communication methods: TCP port mapping and a *custom bridge network*. TCP port mapping, or port forwarding, maps ports from a container to the host, making services running inside a Docker container accessible from the host system and, when necessary, from external networks. Docker also offers custom bridge networks for inter-container communication.

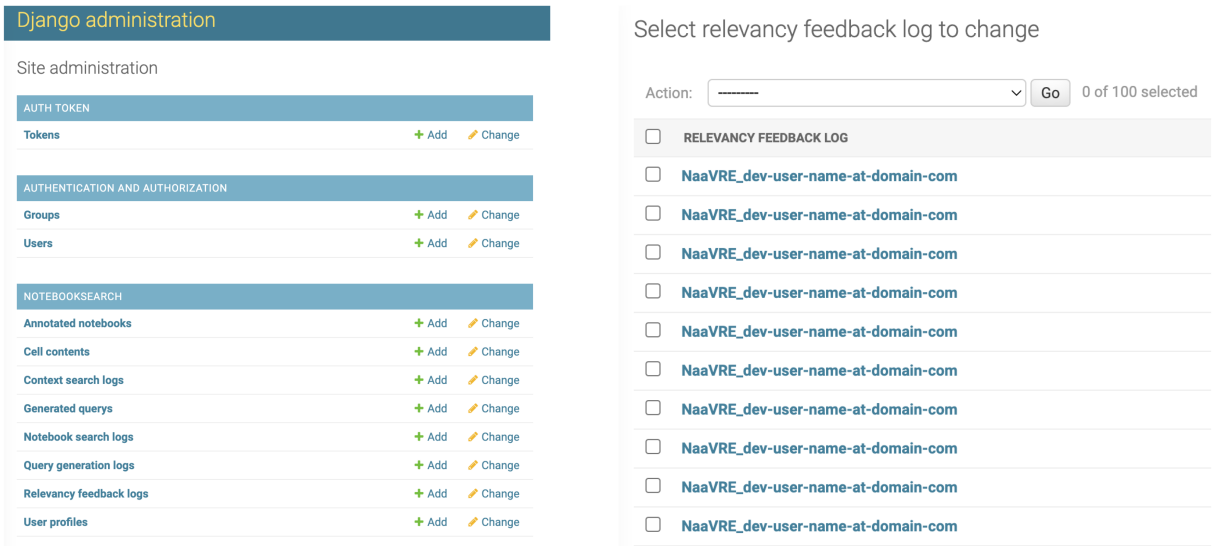
Users access the system through the public IP address of the host. Users' requests are received by port 80 of the host system and redirected to port 80 inside the Nginx container (where a

web server is deployed) via port mapping. For security purposes, port 443 can be employed with the Hypertext Transfer Protocol Secure (HTTPS) protocol. Behind the web server are the application server, the index database, and the user database, each containerized separately. We launch all containers using the *Docker Compose* tool, which creates a custom bridge network to connect them. This ensures that each container can be accessed by other containers on the network using the container name as the hostname and the port number designated for the corresponding servers.

4.4 | Data Collection Mechanism

Research asset search presents a valuable yet challenging research problem that requires ongoing efforts. However, they always encounter significant data challenges, such as the scarcity of comprehensive queries and high-quality relevance assessments, as highlighted in prior research [10]. One approach to address these challenges is through the construction of annotated datasets using crowd-sourcing. However, this method can be time-consuming, expensive, and sometimes unreliable, as participants may not necessarily be the target users—researchers and scientists. Alternatively, a promising strategy is to build datasets dynamically online, gathering user feedback during their interaction with the system. Over time, this approach accumulates queries and relevance judgments from users, resulting in a comprehensive and realistic dataset tailored to specific domains and user expertise.

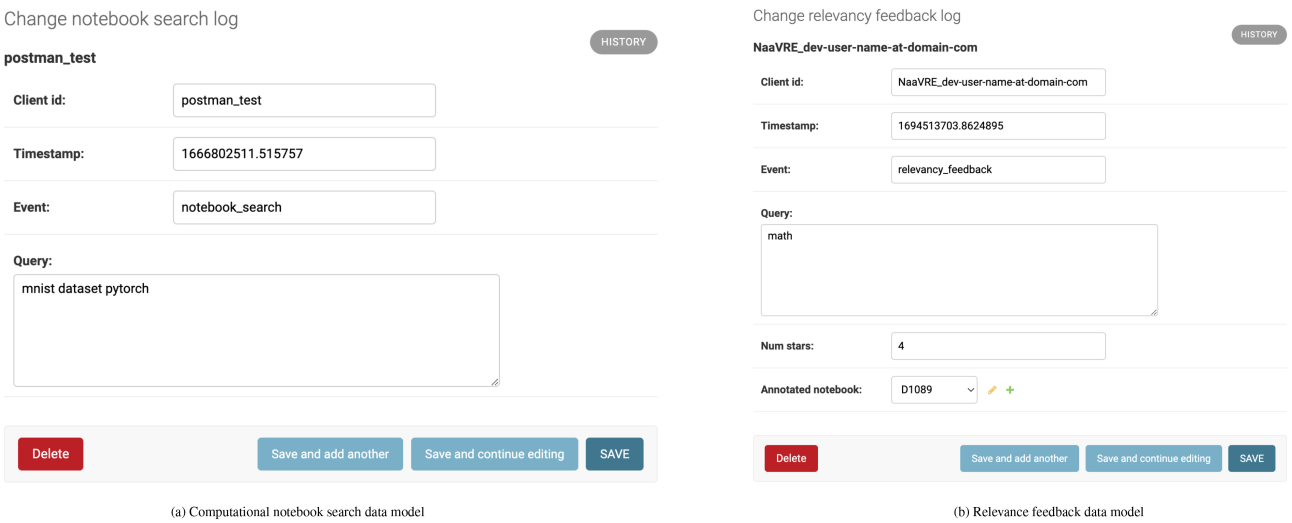
In order to accomplish this objective, we have created several data models that are encompassed by the user database of MRAS, and utilizing the Jupyter interface for collecting data. Figure 7 presents a list of these registered data models within the database, along with the relevance judgments collected from users through the NaaVRE interface. Figure 8 illustrates two data model



(a) Registered data tables

(b) Relevance feedback records

FIGURE 7 | Registered data models for data collection and relevance judgment records.



(a) Computational notebook search data model

(b) Relevance feedback data model

FIGURE 8 | Data models for computational notebook search history and relevance judgments.

examples, that is, *notebook_search* and *relevancy_feedback*. The *notebook_search* table records user search history, capturing details such as client ID, search timestamp, and query content. This information is valuable for analyzing user queries. The relevancy_feedback is designed for gathering relevancy feedback, including fields of the query, the ID of annotated computational notebooks, and the user-rated stars indicating the notebook's usefulness. These human-produced relevancy judgments for query and computational notebook pairs will pile up over time and can be used to expedite computational notebook search studies.

4.5 | Ensuring Implementation Correctness

We use the following methods and tools to ensure the accuracy and correctness of our implementation.

- Unit testing: We divide the system into distinct modules and test each independently. Dedicated testing scripts are developed to validate components such as crawling, preprocessing, indexing, and retrieval.
- API testing: We simulate specific API calls based on predefined data exchange protocols to verify correct implementation. This includes evaluating input/output constraints and ensuring data formats adhere to standards through API responses and server-side state changes.
- Database inspection tools: We employ a variety of tools to monitor and inspect database status, such as availability, index aliases, and the number of stored documents. For example, database indexes are reviewed and validated each time indexed documents are modified.
- Manual semantic-level review: System designers conduct manual checks to ensure the implemented system aligns with its design. This includes validating proper functionality implementation, verifying the relevance of ranking results for specific queries, and ensuring correctness in site navigation and transitions.

4.6 | Summary

Our prototype primarily focuses on achieving key functionalities in a relatively simple environment. To transition from a prototype to a production-ready system, several critical features should be enhanced. First, the system must possess robust error handling and recovery mechanism. In a production environment, the system must be resilient to failures, which demands comprehensive error handling, fault tolerance, automatic recovery, and data backup to ensure the system remains operational in the event of crashes or failures. Second, it is critical to incorporate security features. Production systems must adhere to stringent security practices, such as data encryption, secure authentication/authorization mechanisms, and protection against vulnerabilities like SQL injection or XSS attacks. Lastly, production-ready systems must comply with legal standards such as data privacy laws (GDPR, CCPA) or sector-specific regulations (HIPAA for healthcare, PCI DSS for financial data). This includes data protection, user consent management, and ensuring the system passes regular audits.

5 | Empirical Evidences

The proposed MRAS system is versatile and applicable across many domains. It can be accessed within a Jupyter Notebook environment through our Jupyter integration solution or utilized as a standalone search engine. We demonstrate its usability through three use cases: First, as a standalone search engine in digital pathology within the CLARIFY project and in environmental sciences in collaboration with the ENVRI-FAIR project. Second, through the integration of the MRAS system with Jupyter within the Notebook-as-a-VRE (NaaVRE) framework [4], supporting an effective virtual research environment. For each use case, we created a deployment instance of our MRAS system. Given the diverse architecture of interacting systems, we have three different deployment modes. Figure 9 summarizes the deployment variants under different use cases.

5.1 | Case 1: Digital Pathology

The first use case highlights the application of MRAS in digital pathology, specifically within the CLARIFY project. The success of CLARIFY depends on collaboration among three roles—AI scientists, data scientists, and pathologists—to develop AI solutions for pathological analysis. Given the project's resource constraints and the exploratory nature of research, AI scientists

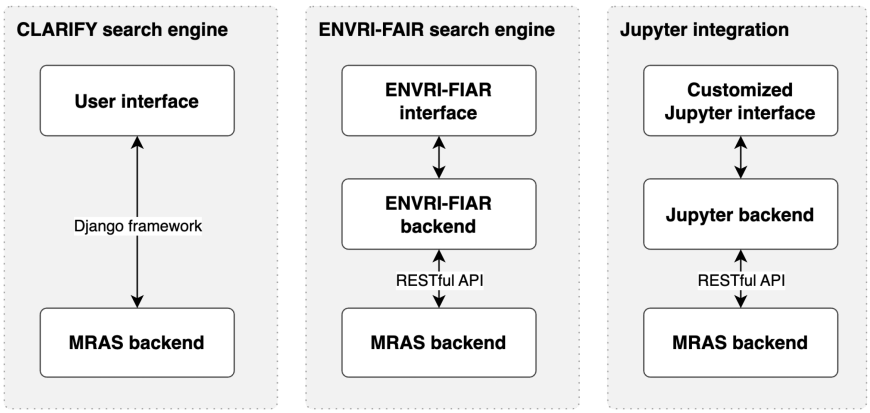


FIGURE 9 | The system deployment variants in three different use cases.

TABLE 1 | Queries collected from researchers studying histopathological image analysis using machine learning methods.

Index	Query	Returning results from Kaggle?
1	Segmentation of epidermis in histopathological images	✗
2	Whole slide images classification TensorFlow code	✓
3	Deep Kernel Learning	✓
4	Mitosis detection in histological images	✗
5	Weakly supervised	✓
6	Prostate gland segmentation via encoder-decoder neural network	✓
7	GAN for image synthesis	✓
8	Conditional GAN for mitosis segmentation	✓
9	Multiple instances learning GitHub code with Pytorch	✓
10	Histological Image Preprocessing	✓

frequently require external research assets to advance their studies. MRAS fulfills this need by assisting researchers in searching computational notebooks related to digital pathology and AI algorithms, thereby accelerating their development process.

To understand users' needs in searching computational notebooks, we designed a questionnaire to collect queries from the partners of the CLARIFY project. Participants were asked to write down at least five natural language queries used to search for codes that are related to their research. We collected 37 queries from seven researchers who work on histopathological image analysis using ML methods. Table 1 lists some examples of the collected queries.

To align the system's knowledge base with the end users' information needs, we utilize the collected queries as keywords to gather computational notebooks in public code repositories from the web. We use Kaggle[8] since it is reported that the most voted computational notebooks published on Kaggle are well-documented [32]. We sent the collected queries to Kaggle and collected a total of 1710 computational notebooks (also called "kernels") using Kaggle API.

For some queries, Kaggle did not return any computational notebooks, as detailed in Table 1. The statistics of the collected notebooks are presented in Table 2. Notably, 99.6% of the notebooks are written in Python. The average and median numbers of code cells and Markdown cells are comparable. However, the average and median text length (measured by the number of characters) within Markdown cells vary significantly, with the average length being 17,358 characters and the median length only 6252 characters. To expand the search space, we included computational notebooks retrieved using other common keywords. Each computational notebook is accompanied by

TABLE 2 | Distributions of collected computational notebooks from Kaggle.

	# code cells	# MD cells	Len. MD text
Average	33	25	17,358
Median	24	17	6252
Standard Deviation	31	27	54,518

Abbreviation: MD: Markdown.

metadata describing contextual information such as "id," "title," "language," "kernel_type," "dataset_sources," and more.

Given an aligned search space with researchers' interests, we provide CLARIFY partners with a customized search system for computational notebook search within the digital pathology domain. Figure 10 illustrates the SERP of returned computational notebooks using the query "cell segmentation." The computational notebooks are ranked by the relevance to the query. When users click on one item, the webpage will be directed to the source page of the computational notebook, where users can acquire more contextual information and download the source files.

5.2 | Case 2: Environmental Science

In the second use case, we illustrate the application of our system in the environmental science domain through our collaboration with the EU-funded ENVRI-FAIR project. The European Environmental and Earth System Research Infrastructure (ENVRI) is a community of environmental Research infrastructures working together to observe the Earth as one system. It provides high-quality digital assets like research data and services. The ENVRI-FAIR project aims to implement the findability, accessibility, interoperability, and reusability (FAIRness) of these assets in the ENVRI community and connect them to the European Open Science Cloud (EOSC).

Unlike the solutions provided to CLARIFY, ENVRI-FAIR has its own search engine[9] for discovering resources in environmental science, including web pages, datasets, web APIs, and images. However, it lacked the capability to search for computational notebooks. To address this, they collaborated with us to integrate a computational notebook search feature specifically for environmental science. This integration is enabled by RESTful APIs provided by the MRAS system. The MRAS system is first deployed as a web application with full-ledge RESTful APIs, By calling these APIs, the ENVRI-FAIR search platform can forward users' queries to the MRAS system and incorporate the ranked results into its search results page. Figure 11 shows the computational notebook search results within the ENVRI-FAIR search engine.

5.3 | Case 3: Integration With Jupyter Notebook

The third use case demonstrates the integration of our system with a Jupyter Notebook. Notebook programming environments, such as the Jupyter Notebook, receive increasing popularity among data scientists to prototype and execute computational experiments attributed to their interactivity

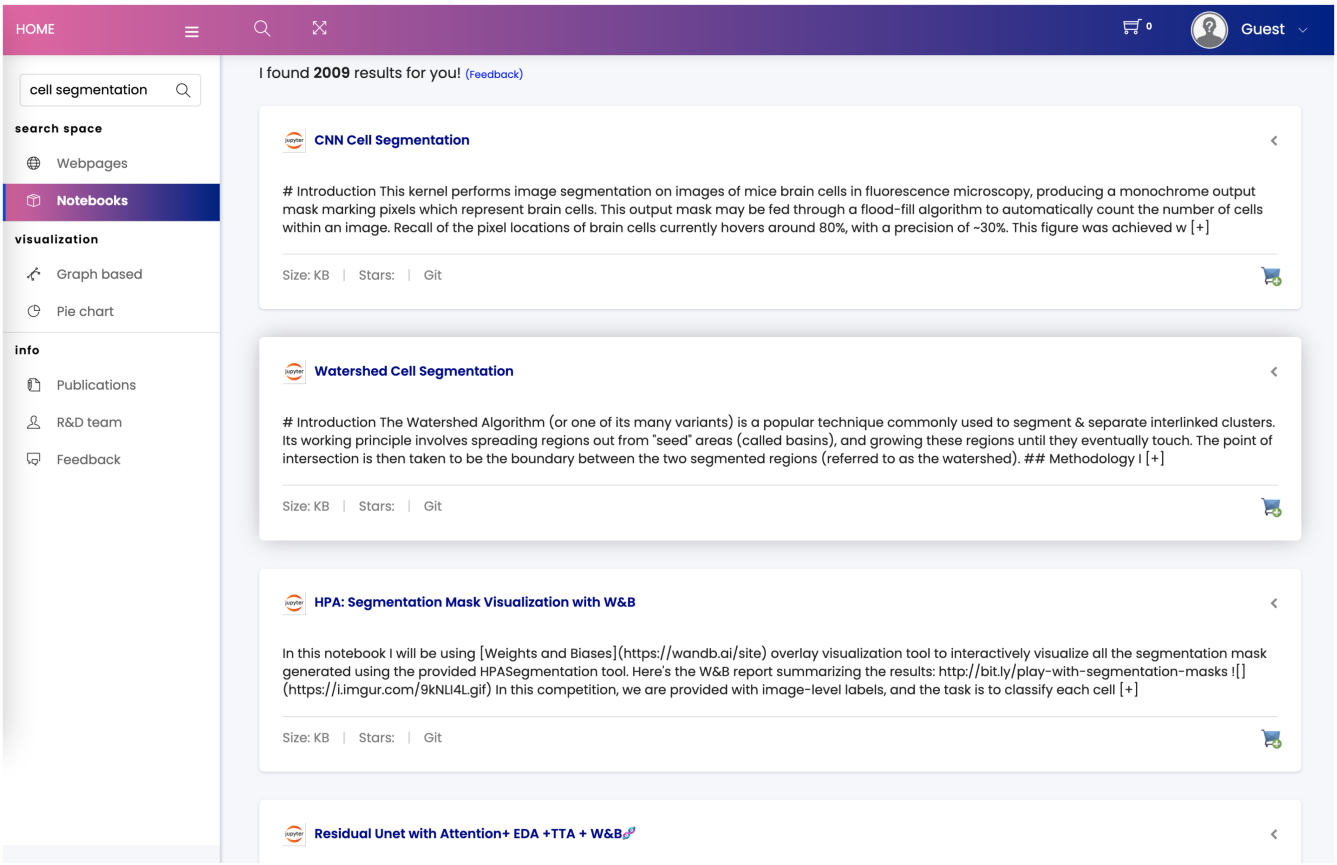


FIGURE 10 | The computational notebook search results in the CLARIFY search engine.

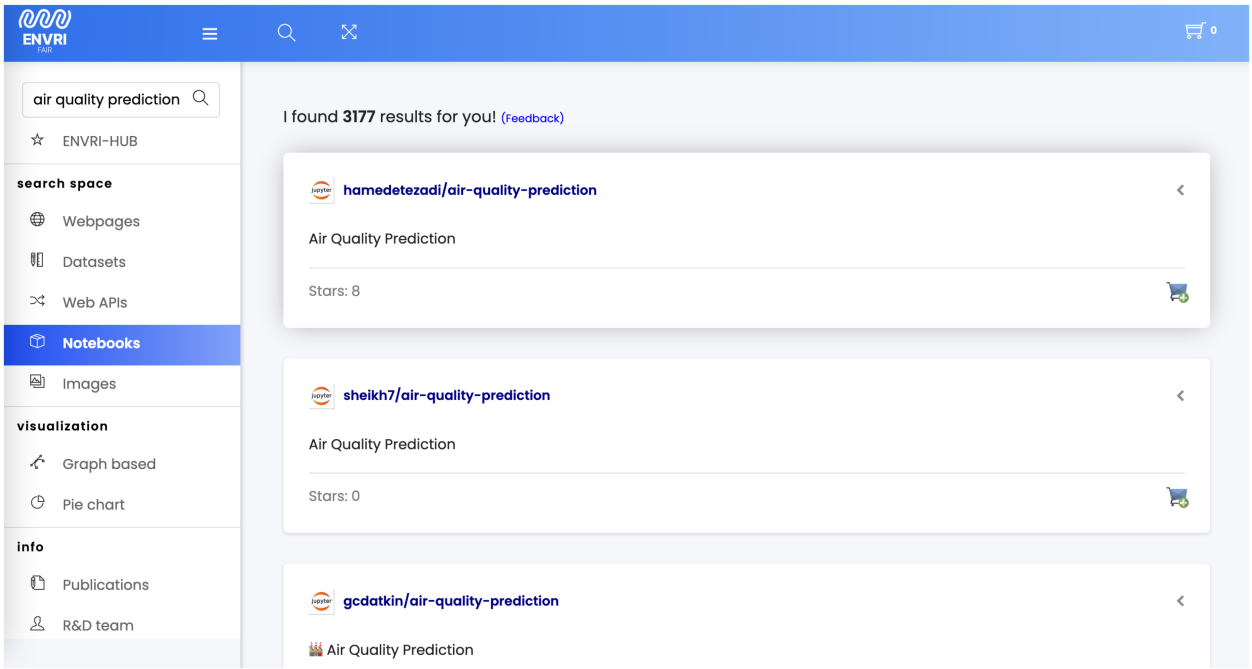


FIGURE 11 | The computational notebook search results in the ENVRI-FAIR search engine.

and flexibility. NaaVRE utilizes Jupyter Notebook to create a cloud-based collaborative platform for scientists to carry out research activities [4]. Integrating search functions for research assets can significantly enhance the efficiency of finding and reusing existing assets in the research workflow composition process.

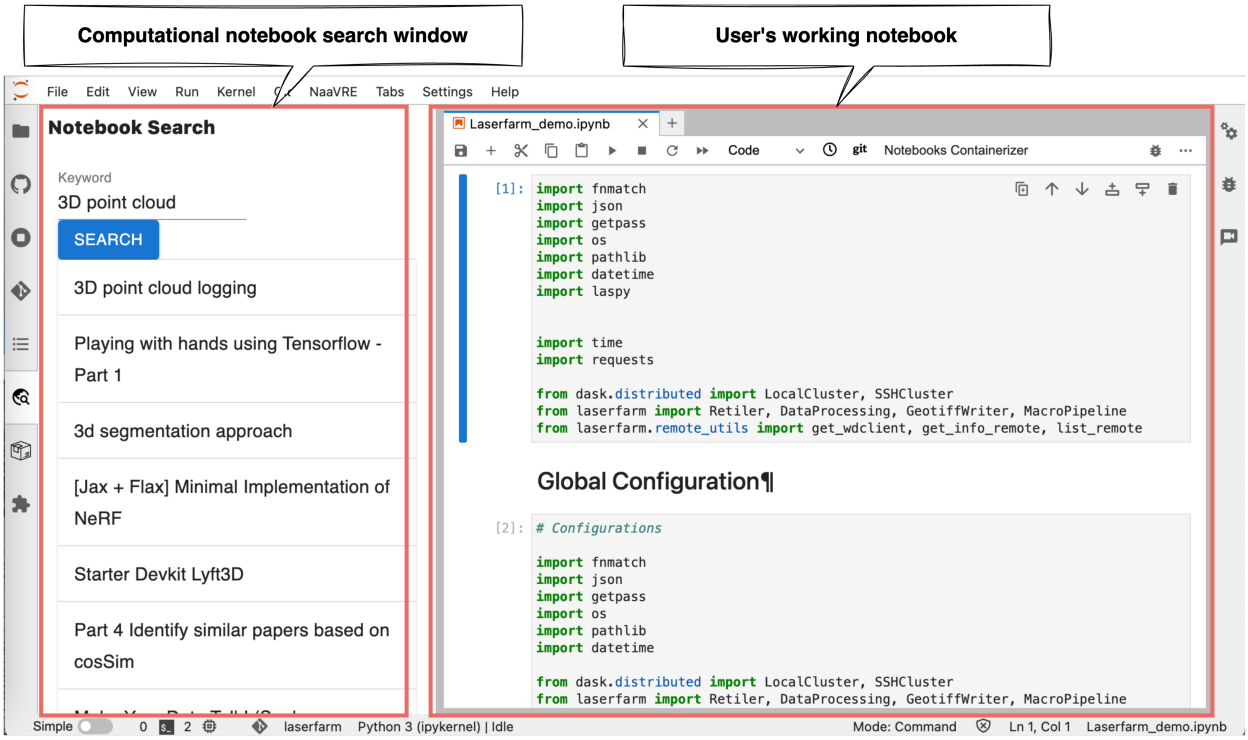


FIGURE 12 | The user interface for computational notebook search in NaaVRE.

Since Jupyter Notebook is a web application with both a frontend interface and a backend server, our integration strategy is similar to the one used for the ENVRI-FAIR search platform. Specifically, we modify the Jupyter Notebook kernel to manage the application logic and the communication with the MRAS system through API calls. For integration with the NaaVRE framework, the MRAS system is deployed as a web application with comprehensive RESTful APIs that interact with the Jupyter backend via API calls. However, the Jupyter Notebook interface does not naturally support search functionality. To address this, we customize the Notebook interface to include search bars and result pages and establish corresponding data transfer channels between the interactive elements and the backend data variables.

Figure 12 displays NaaVRE's computational notebook search interface, functioning as an extension within the Jupyter Notebook environment. By clicking the search icon in the sidebar, a search window appears adjacent to the user's active notebook, enabling real-time searches across external computational notebooks. Users can navigate through search results and click on items to obtain detailed information. As depicted in Figure 13, users can examine the raw content of these computational notebooks to determine their relevance. If a computational notebook proves useful, users can easily download its source file to their workspace, establish a Jupyter kernel for executing code, and seamlessly integrate it with other computational notebooks. This is possible because all the computational notebooks within the search space have been downloaded and stored in the MRAS system's database beforehand, such that they can be directly fetched with document identities during the search process. Moreover, the Jupyter interface incorporates a rating system for assessed computational notebooks. Ratings utilize a five-star scale, with higher ratings indicating greater utility for the current task. These ratings are transmitted to the computational notebook search system's backend and stored in the user database. This data contributes to refining the search engine's ranking algorithms, thereby enhancing its effectiveness over time.

6 | Results and Analysis

6.1 | Requirement Fulfillment

The proposed MRAS system effectively meets all outlined requirements, serving as an efficient solution for discovering a wide range of research materials.

We developed specialized preprocessing and indexing pipelines to unify datasets and computational notebooks sourced from diverse formats and origins. These consolidated indexes are integrated into our central knowledge base, providing a unified access point for searching diverse resources. This integration significantly enhances researcher productivity by enabling simultaneous exploration of multiple relevant assets. The system prioritizes effective ranking and swift retrieval of research assets. While advanced algorithms like dense retrieval are available, we chose BM25 and inverted index technologies for their superior efficiency, albeit with a slight trade-off in ranking precision. MRAS also performs content analysis specifically tailored for computational notebooks, which often contain detailed task descriptions, methodologies, and key findings in Markdown cells. Extracting and enriching this information in our indexes allows users to quickly understand each computational notebook's purpose and content, facilitating informed decision-making. Seamless integration with Jupyter Notebook is another prominent feature of MRAS. By extending NaaVRE

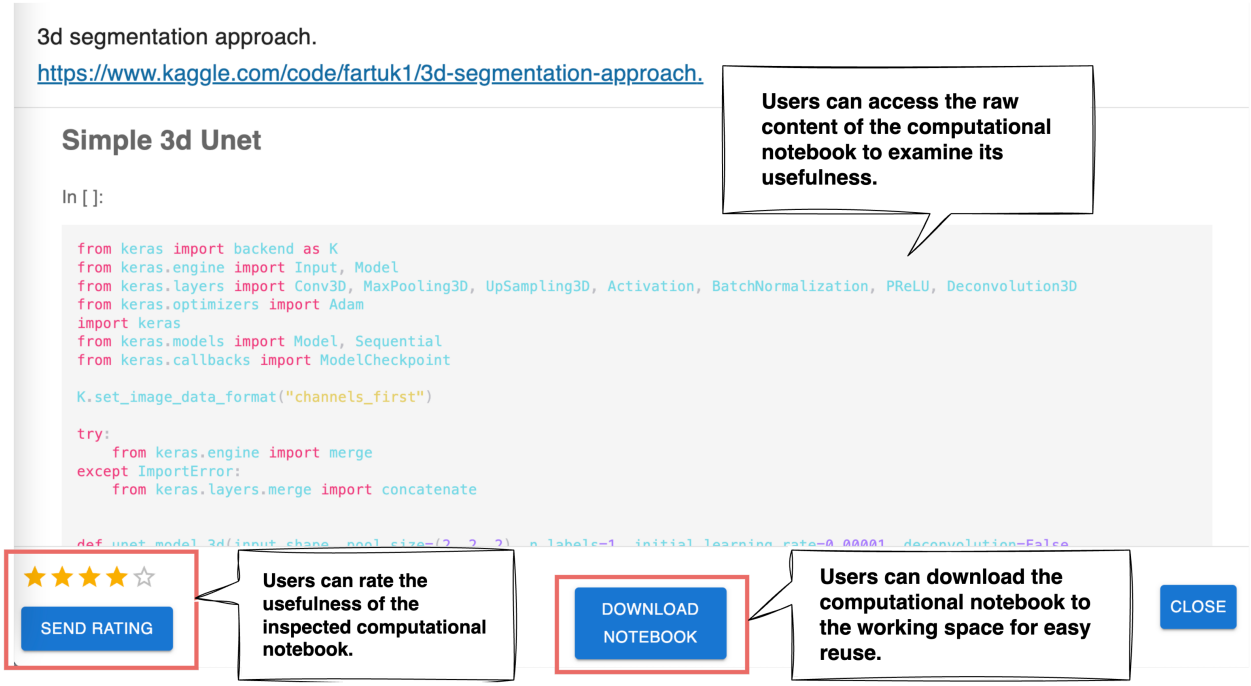


FIGURE 13 | Inspecting a retrieved computational notebook.

with MRAS's computational notebook search capabilities, users experience a smooth transition between external research asset discovery and their research environment.

Addressing nonfunctional requirements, MRAS excels in presenting comprehensive metadata for each research asset, including names, sizes, and data source details. This transparency enables users to evaluate resource quality and suitability effectively. Additionally, the system incorporates user feedback mechanisms such as search logs and relevance feedback, empowering users to influence search result rankings and continuously enhance system relevance and effectiveness based on analyzed interactions.

6.2 | System Responsiveness

We deployed the system on a virtual machine with an 8-core CPU, 8GB memory, and 64GB disk storage capacity to evaluate its responsiveness. To simulate multiple users, we sent 30 distinct queries simultaneously to the *notebook search* API, which handles the queries and returns ranked computational notebooks. This process was repeated for five rounds to minimize variability in the testing procedure. Figure 14 shows the latency of the API calls in box-plots. The network latency is around 0.02 s, which is negligible compared to the latency caused by system processing overhead. On average, the API responded to all users in under 0.25 s, demonstrating strong responsiveness.

6.3 | Usability Test

We conducted a usability test for the proposed MRAS system. We recruited five active researchers (two female and three male) from different fields. Their ages are between 23 and 38. Their background includes data science, computer science, AI, and cloud computing, while their research interests include IR; electronic design automation; ML; optimization and intelligent network; anomaly detection, workflow adaptation, digital twin.

We ask the participants to finish two tasks using our system and fill in a questionnaire targeting their usage experiences of the system. Task 1 requires them to find a desired computational notebook while task 2 a desired dataset. The concrete instructions are as follows:

1. Use queries related to your research/project to search for notebooks.
2. Go through the returned results and find the most desired notebooks for your needs.
3. Change your query if necessary.
4. Switch to the datasets panel (left part) and search for datasets.
5. Go through the returned results and find the most desired datasets for your needs.
6. Change your query if necessary.

6.3.1 | Search Effectiveness

In the computational notebook search, two participants utilized two queries, while three participants employed three queries. Four participants successfully located the computational notebooks they were searching for, while one was unable to find the intended ones. For the dataset search, one participant used a single query, three participants used three queries, and one participant used six queries. Ultimately, three participants successfully

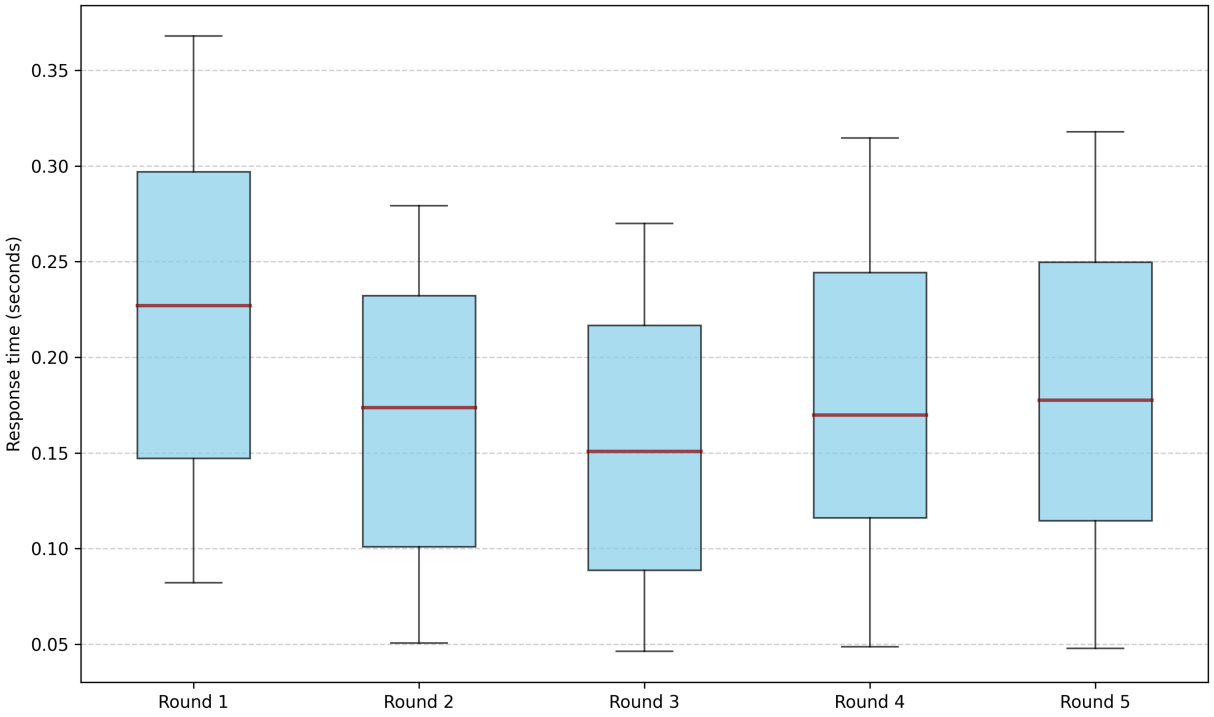


FIGURE 14 | The latency of API calls of the deployed system.

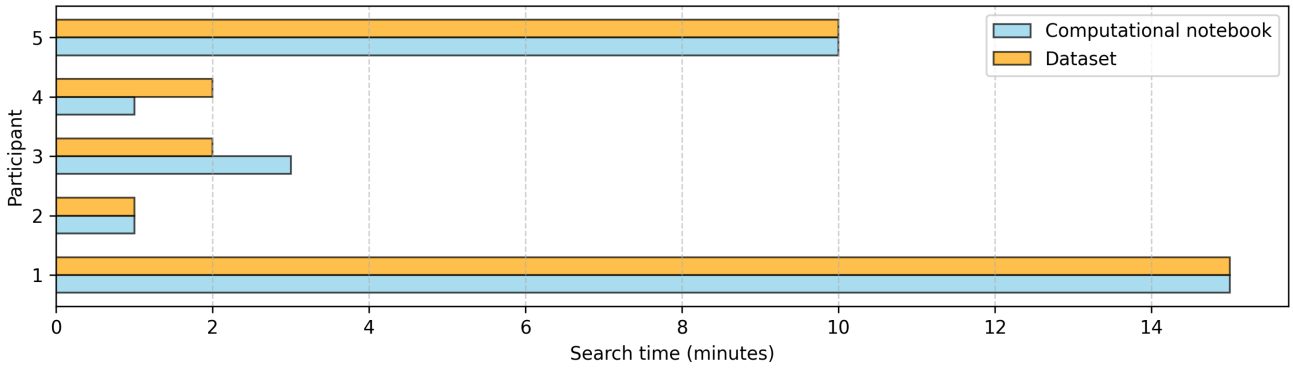


FIGURE 15 | Reported time spent on tasks.

found their desired datasets, whereas two were unsuccessful. Participants reported the time they spent on finishing each task, as depicted in Figure 15. The time spent on tasks varied significantly. While some participants completed each task relatively quickly, in just 2 or 3 min, others took much longer, spending 10–15 min on both tasks.

We ask them for the possible reasons behind the failure to discover desired research assets. The results reported by participants are summarized in Table 3. These results highlight some key factors of an effective search system. These include an appropriate search space, sufficient result display, and an advanced query understanding module that may assist in users' query reformulation.

6.3.2 | Result Presentation

We assess the usefulness of descriptions and metadata in the result presentation by asking participants to answer two questions with a 5-scale score (1–5, higher meaning more useful):

- How much do you think the descriptions help you to judge the relevance of the computational notebooks/datasets?
- How much do you think the metadata helps you to judge the relevance of the computational notebooks/datasets?

The average usefulness score for the description of computational notebooks is 4.2, and the metadata is 4.4, whereas the average usefulness score for both the description and the metadata of datasets is 4.2. These scores indicate a high usefulness of the metadata in judging the relevance of returned research assets.

6.3.3 | User-Friendliness

We asked participants to scale 1–5 for the following statements:

- I found the system unnecessarily complex.

TABLE 3 | Reasons provided by participants regarding the inadequacy of search results.

Reasons for computational notebooks	Reasons for datasets
The result page doesn't provide sufficient information.	I can't formulate a proper query to find my desired contents.
The search space doesn't cover my topic.	The search space doesn't cover my topic.
The search space doesn't cover my topic.	The result page doesn't provide sufficient information.
I partially found relevant notebooks.	I partially found relevant datasets.
	I believe the development trends of datasets and research directions are closely related to their level of community involvement. Some are relatively new, some are more commercially oriented, and others have lower popularity within the open-source community.

- I would imagine that most people would learn to use this system very quickly.

The average score for the first statement is 1.8, meaning that the system is not over-complex for its intended purpose. The average score for the second statement rated by participants is 4.4, suggesting their positive attitude toward the user-friendliness of our system.

7 | Discussion

This work focuses on addressing four primary research questions. First, to extract and integrate diverse research assets from multiple sources, we developed a data extraction and indexing pipeline to collect data from various providers and used a central knowledge base to merge asset indexes and user data. Second, to effectively rank research assets, we reviewed related studies on asset retrieval methods and utilized indexing tools to enhance ranking efficiency and effectiveness. Third, for representing computational notebooks, we performed content analysis to extract key information and explored various representation methods. Lastly, we integrated the proposed MRAS system into the Jupyter Notebook environment to support seamless research workflows.

The usability tests are conducted with a small set of users, and it can limit the generalizability of the results, as the findings might not fully represent the experiences or behaviors of a larger, more diverse audience. However, small-scale usability tests are still valuable for identifying major usability issues and trends. These tests often follow the principle that a few users can uncover the most critical problems, but additional testing with larger or more varied groups may be needed for comprehensive insights.

We provide some lessons learned from this study. First, gathering research assets is crucial. Research assets, such as datasets and computational notebooks constitute a small fraction of web content. One way is crawling web pages and filtering research assets from enormous web pages. This requires advanced crawling techniques and web markup standards such as Schema.org.[10] Alternatively, we can collect a list of data sources, such as repositories, catalogs, and data portals from the community, and build customized crawlers for registered data sources. This approach is easy to implement and provides community-recognized resources. However, it depends on active community participation and might miss less popular sources. Secondly, it is important to properly define the offline and the online processes to improve the search system's usability. The online components handle real-time user interactions, whereas the offline components gather, process, and prepare data for online services. For instance, conducting indexing during online operations can introduce significant delays in returning search results, which can impair the user experience.

We also share some practical challenges encountered during the system implementation process.

Request limitation from data sources. Kaggle and GitHub, which serve as primary sources for acquiring computational notebooks, enforce strict API request limits. Consequently, we are compelled to download notebooks in a largely sequential manner, leading to significant time consumption during the collection process. In addition, to mitigate issues caused by request blocking, we employed a "sleep-and-restart" strategy to resume downloads once access restrictions were triggered.

Irrelevant content in computational notebooks. Computational notebooks sometimes contain excessive content, such as embedded HTML, URLs, and static images that clutter the document without contributing value. This can significantly increase the size of the document and adversely affect the indexing and ranking process. Therefore, it is essential to clean the computational notebooks prior to further use.

8 | Conclusion and Future Work

With the growing demand for the reuse of research assets like datasets and codes, there is a need for a system that offers search capabilities for various research resources. However, commonly used search tools, such as general-purpose search engines like Google Search, or code and dataset repositories like Kaggle and Github, are inadequate in meeting the specific search requirements of researchers. To bridge this gap, we propose a MRAS system. This system allows users to search for diverse research assets to support scientific research activities throughout the entire data science lifecycle.

In this article, we focus on two important categories of research assets, that is, computational notebooks and datasets. We reviewed the state of arts pertinent to computational notebook search and dataset search and identified the gaps. We analyzed the requirements, both functional and nonfunctional, for building an effective MRAS system. We introduced the conceptual framework of the proposed system and provided great details

on the implementation of the system, disclosing the underlying technologies of the main building blocks, the data processing process, and the data collection mechanism. We demonstrated the system's usefulness through three different use scenarios: as a full-stack search engine for histopathological image analysis; embedded with external search engines via API services; and integrated with a Jupyter environment for efficient in-site research asset discovery and reuse.

Search behavior occurs frequently at multiple stages in the research lifecycle. The presented MRAS system can significantly enhance the efficiency of research activities, particularly those involved in constructing a data analytic pipeline. By providing a central search platform, our system can save researchers valuable time in finding relevant research assets across different platforms and services. Additionally, our study promotes the reusability of research resources. With increased search efficiency, researchers are encouraged to use existing resources instead of starting from scratch for new projects. This not only reduces redundant efforts but also supports the accumulation of knowledge over time. As the MRAS system evolves and gains traction within the research community, it has the potential for long-term impact. By continuously improving the discovery quality of research assets and fostering a culture of resource sharing and reuse, our work can contribute to the sustainability and growth of scientific research.

Throughout the research journey of building a MRAS system, we have come to realize that tackling a MRAS challenge goes beyond just a IR problem. The primary goal of such a search is to compile various research assets into a cohesive workflow to address scientific questions. This requires evaluating the usability of the acquired resources and ensuring their compatibility with different types of research assets. These insights highlight the open issues of the research on MRAS. First, quantitative measurements and metrics need to be established for assessing the quality and reusability of research assets, which should be integrated into retrieval methods to rank resources more effectively. For example, the quality of a dataset can be evaluated based on the completeness of its documentation, the reliability of its data source, and internal quality indicators such as anomaly, consistency, bias, and fairness. Similarly, the quality of computational notebooks can be assessed through the clarity of the textual description, the readability and correctness of the code, and the documentation of data sources and dependencies. Second, it is essential to map out how different types of research assets are related. For example, showing that dataset A is utilized by computational notebook B or that they are both used for the same task. Understanding these connections will help researchers select the best combination of resources for their specific research objectives, ultimately improving the efficiency of their work.

In line with Open Science principles, the source code will be made accessible on public platforms like GitHub. Our research team includes dedicated software developers who will manage and improve the system. Additionally, the system will continue to evolve as new research findings emerge.

Author Contributions

All authors contributed to the conceptualization and writing of the paper; more specifically, Na Li led the development and deployment of the MRAS system, conducted case studies, and contributed to the writing and revision of the paper. Siamak Farshidi contributed to the structure design and revision of the paper. Zhiming Zhao contributed to the design of the MRAS system, coordinated the case studies, and contributed to the writing and revision of the paper.

Acknowledgments

This work has been partially funded by the EU H2020 CLARIFY project (860627), EU LifeWatch ERIC, Dutch Research Council, LTER-LIFE project, EU Horizon Europe BLUECLOUD 2026 project (101094227), and EU Horizon-Europe EVERSE project (101129744), EU Horizon-Europe ENVRI-HUB next project (101131141).

Conflicts of Interest

The authors declare no conflicts of interest.

Data Availability Statement

The data that support the findings of this study are openly available in research_asset_search at https://github.com/nali001/research_asset_search.

Endnotes

[1] https://colab.research.google.com/.

[2] https://www.kaggle.com/code.

[3] https://www.google.com/.

[4] https://www.bing.com/.

[5] https://data.icos-cp.eu/portal/.

[6] https://metadatacatalogue.lifewatch.eu/.

[7] https://zenodo.org/.

[8] https://www.kaggle.com/.

[9] https://search.envri.eu/.

[10] https://schema.org/.

References

1. A. Madabhushi and G. Lee, "Image Analysis and Machine Learning in Digital Pathology: Challenges and Opportunities," *Medical Image Analysis* 33 (2016): 170–175.

2. M. Scowen, I. N. Athanasiadis, J. M. Bullock, F. Eigenbrod, and S. Willcock, "The Current and Future Uses of Machine Learning in Ecosystem Service Research," *Science of the Total Environment* 799 (2021): 149263.

3. A. Arunarani, D. Manjula, and V. Sugumaran, "Task Scheduling Techniques in Cloud Computing: A Literature Survey," *Future Generation Computer Systems* 91 (2019): 407–415.

4. Z. Zhao, S. Koulouzis, R. Bianchi, et al., "Notebook-as-a-VRE (NaaVRE): From Private Notebooks to a Collaborative Cloud Virtual Research Environment," *Software: Practice and Experience* 52, no. 9 (2022): 1947–1966.

5. D. Brickley, M. Burgess, and N. Noy, "Google Dataset Search: Building a Search Engine for Datasets in an Open Web Ecosystem," in *WWW '19: The World Wide Web Conference* (New York, NY, USA: Association for Computing Machinery, 2019), 1365–1375.

6. Q. Huang, Y. Yang, X. Zhan, H. Wan, and G. Wu, "Query Expansion Based on Statistical Learning From Code Changes," *Software: Practice and Experience* 48, no. 7 (2018): 1333–1351.

7. G. Hu, M. Peng, Y. Zhang, Q. Xie, W. Gao, and M. Yuan, "Unsupervised Software Repositories Mining and Its Application to Code Search," *Software: Practice and Experience* 50, no. 3 (2020): 299–322.

8. Z. Zhou, H. Yu, and G. Fan, "Effective Approaches to Combining Lexical and Syntactical Information for Code Summarization," *Software: Practice and Experience* 50, no. 12 (2020): 2313–2336.

9. J. Wonsil, N. Boufford, P. Agrawal, et al., "Reproducibility as a Service," *Software: Practice and Experience* 53, no. 7 (2023): 1543–1571.

10. N. Li, Y. Zhang, and Z. Zhao, "A Dense Retrieval System and Evaluation Dataset for Scientific Computational Notebooks," in *2023 IEEE 19th International Conference on e-Science (e-Science)* (2023), 1–10.

11. N. Li, Y. Zhang, and Z. Zhao, "CNSVRE: A Query Reformulated Search System With Explainable Summarization for Virtual Research Environment," in *WWW '23 Companion: Companion Proceedings of the ACM Web Conference 2023* (New York, NY, USA: Association for Computing Machinery, 2023), 254–257.

12. S. Robertson and H. Zaragoza, *The Probabilistic Relevance Framework: BM25 and Beyond* (Hanover, MA, USA: Now Publishers Inc., 2009), 333–389.

13. N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing* (Hongkong, China: Association for Computational Linguistics, 2019), 3982–3992.

14. X. Li, Y. Wang, H. Wang, Y. Wang, and J. Zhao, "NBSearch: Semantic Search and Visual Exploration of Computational Notebooks," in *CHI '21: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA: Association for Computing Machinery, 2021), 1–14.

15. M. Horiuchi, Y. Sasaki, C. Xiao, and M. Onizuka, "JupySim: Jupyter Notebook Similarity Search System," *Open Proceedings* (Edinburgh, UK: EDBT, 2022).

16. 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* 13, no. 1 (2023): 1–27.

17. S. Castelo, R. Rampin, A. Santos, A. Bessa, F. Chirigati, and J. Freire, "Auctus: A Dataset Search Engine for Data Discovery and Augmentation," *Proceedings of the VLDB Endowment* 14, no. 12 (2021): 2791–2794.

18. 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 (2021): 329–345.

19. J. Ramos, "Using TF-IDF to Determine Word Relevance in Document Queries," *Proceedings 1st Instructional Conference on Machine Learning* 242 (2003): 133–142.

20. P. S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck, "Learning Deep Structured Semantic Models for Web Search Using Clickthrough Data," in *CIKM '13: Proceedings of the 22nd ACM International Conference on Information & Knowledge Management* (New York, NY, USA: Association for Computing Machinery, 2013), 2333–2338.

21. V. Karpukhin, B. Oguz, S. Min, et al., "Dense Passage Retrieval for Open-Domain Question Answering," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (Association for Computational Linguistics, 2020), 6769–6781.

22. O. Khattab and M. Zaharia, "ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction Over BERT," in *SIGIR '20: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA: Association for Computing Machinery, 2020), 39–48.

23. C. Liu, X. Xia, D. Lo, C. Gao, X. Yang, and J. Grundy, "Opportunities and Challenges in Code Search Tools," *ACM Computing Surveys (CSUR)* 54, no. 9 (2021): 1–40.

24. F. Lv, H. Zhang, L. J-g, 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)* (Lincoln, NE: IEEE, 2015), 260–270.

25. S. Sachdev, H. Li, S. Luan, S. Kim, K. Sen, and S. Chandra, "Retrieval on Source Code: A Neural Code Search," in *MAPL 2018: Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages* (New York, NY, USA: Association for Computing Machinery, 2018), 31–41.

26. X. Ling, L. Wu, S. Wang, et al., "Deep Graph Matching and Searching for Semantic Code Retrieval," *ACM Transactions on Knowledge Discovery From Data (TKDD)* 15, no. 5 (2021): 1–21.

27. C. Zeng, Y. Yu, S. Li, et al., "deGraphCS: Embedding Variable-Based Flow Graph for Neural Code Search," *ACM Transactions on Software Engineering and Methodology* 32, no. 2 (2023): 1–27.

28. X. Gu, H. Zhang, and S. Kim, "Deep Code Search," in *ICSE '18: Proceedings of the 40th International Conference on Software Engineering* (New York, NY, USA: Association for Computing Machinery, 2018), 933–944.

29. A. Chapman, E. Simperl, L. Koesten, et al., "Dataset Search: A Survey," *VLDB Journal* 29, no. 1 (2020): 251–272.

30. C. Raffel, N. Shazeer, A. Roberts, et al., "Exploring the Limits of Transfer Learning With a Unified Text-To-Text Transformer," *Journal of Machine Learning Research* 21, no. 1 (2020): 5485–5551.

31. S. Farshidi and Z. Zhao, "An Adaptable Indexing Pipeline for Enriching Meta Information of Datasets From Heterogeneous Repositories," in *Advances in Knowledge Discovery and Data Mining. PAKDD 2022* (Cham: Springer, 2022), 472–484.

32. A. Y. Wang, D. Wang, J. Drozdal, et al., "What Makes a Well-Documented Notebook? A Case Study of Data Scientists' Documentation Practices in Kaggle," in *CHI EA '21: Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA: Association for Computing Machinery, 2021), 1–7.