Building a user-friendly information retrieval system to facilitate medical search

(Please contact leonardo.castorina@ed.ac.uk – s2090655 – to organize a meeting for a live demo)

I. Introduction

The world wide web has become a primary source of information across many domains with an exponential growth in both volume and number of entries available. Among these, health, medical and related biology information is one of the leading domains, with 83% of adult internet users searching online for health information [2]. This can be attributed to people becoming more health conscious in general as well as patients wanting to be more directly involved in the decision-making process both before, during and after consulting with physicians. Therefore, they look to the web in order to arm themselves with knowledge. Correspondingly, the first step for any good physician, medical professional or researcher is to be well versed in the literature of the subject as they are faced with more nuanced questions and are expected to provide up-to-date, comprehensive answers and solutions. This requires easy and fast access to information and sources related to their inquiry which can satisfy the diverse needs of users which range from the layperson with limited to no medical background to the medical physician already equipped with years of training.

For example, a clinician who represents a typical example of an expert medical user, may use a web search engine (SE) to assist them in solving a difficult case, such as a rare disease. The professional might perform a diagnostic health search online as a coarse form of hypothetico-deductive reasoning [4], using evidence from a SE to guide an iterative cycle of hypotheses about the disease. In this case, more detailed, scientific search results may be preferred. On the other hand, a patient may express specific symptoms they would like to inquire about or have been diagnosed with a particular disorder and would like to learn more about it in preparation for additional visits to the hospital.

A recent study on how people consume health information on the web showed that people tend to start medical searches on general SEs as opposed to medical libraries [1]. However, information retrieved from scientific sites and public institutions originating from such engines are typically considered more trustworthy and relevant for those who want more detail beyond simply WebMD, for instance. However, this must be balanced against the fact that information with overly heavy medical terminology is also not useful given some subjects may continue their search even after finding a page with the answer due to not being able to understand the information presented. These two scenarios highlight the growing need for medical and biological information retrieval tools for search activity, particularly exploratory health search [8] which goes beyond simple fact retrieval and is aimed at both expert and non-expert users.

A. System Proposal Highlights

Motivated by these observations, we developed MediSia ("Sia" representing the Egyptian god of wisdom), a SE specifically designed to retrieve medical and biology-related literature from the internet with a user-friendly interface for both layperson and professional use. It uses a specially curated dataset which was crawled from freely available online resources, consisting of both journal articles as well as review papers (which may particularly be of greater interest to non-experts looking for summarized facts and findings). MediSia's advantage is not being limited to a specific medical information collection but can be used by anybody who wishes to perform a simple yet tailored literature search about particular health information. This enables MediSia to supply information from a repository of specialized,

"clean" resources as opposed to general web SEs which search the entire web and are more likely to return commercial, erroneous and less relevant results.

In summary, our system is equipped to take a user's query and match this query to a topic (or a set of topics) in medical terminology. The system then ranks the query results according to document relevance. By exploiting information retrieval techniques such as TF-IDF and cosine similarity, MediSia is able to accurately filter irrelevant documents while also enabling the user flexibility to provide more or less detail based on their personal knowledge or background. In this paper, we present MediSia including details regarding its function, design and capabilities. The code and data for this system are available at https://github.com/universvm/MediSia. Given the large size of the dataset, this search engine is currently hosted on a private machine in the School of Informatics. The contributions of this project are summarized below:

- A new SE system used to retrieve medical and biological literature-based information available on the Internet across multiple sources for both users with and without previous medical or scientific training
- A system that is capable of taking a simple (or complex) query specification depending on the user's background and match it to a topic or set of topics commonly used in the medical and biological fields for faster processing and search speed
- A literature retrieval database powered by a topical web crawler and an indexing strategy that exploits the advantage of medical and biology domain knowledge to filter irrelevant search results

The paper is organized as follows: Section 2 discusses the database collection and automated filtering process for evaluating journal articles, reviews and other relevant literature. Section 3 begins our presentation of MediSia's structure starting with our indexing approach including preprocessing and text transformation details as well as our unique multi-indexing topical approach. Section 4 presents our search retrieval model design, including our novel query classification design for efficiency, multiple search functionality options and our ranked retrieval approach. Section 5 presents our frontend web interface designed to provide a user-friendly layout and describes the user input including the range of search parameters available and additional follow-up search options. Section 6 summarizes our learnings, challenges and possible future directions for potential improvement and scale. Finally, Section 7 provides details regarding our project management and organization including our timeline of events and individual contribution.

II. Data Collection Evaluation

Standard search technology is currently inadequate for the task of literature retrieval on the topic of biology and medical research. While skepticism about the credibility and level of medical information on the internet has motivated the development of specialized medical web portals, such as Orphanet or Phenomizer, these are largely inaccessible to the general public or lack user-friendly features for a non-expert user. On the other hand, general purpose search engines tailored toward public use, such as a Google Search, cover the entirety of web results which is disadvantaged given it often contains noise and non-relevant results (e.g., pages from non-authoritative sources or sponsored content). In order to fill this gap of performing search tasks inside a domain of expertise, MediSia is proposed as a specialized SE which can decrease search time and improve performance.

A. Database Extraction

Unpaywall is an open database of over 28 million free scholarly articles harvested across over 50,000 journals including those commonly seen in peer-reviewed journals such as Science or PLOS One as well as from preprint repositories such as arXiv [10]. This made up our initial data source from which refining and optimizing steps were then taken, including filtering only relevant results and extracting each article's respective abstract using a web crawler across other websites. Unpaywall is formatted such that it contains details of each Crossref DOI as a JSON-formatted data file with the latest version weighing at 124GB. Given the sheer volume of the raw database and thus memory needed, filtering was first critical to limit its size to content related only to medicine and biology.

The filtering process was done in two steps: (1) based on key words in titles and (2) by journal category. First, papers which had titles containing the word "bio*" or "med*" and not "biog*" (to avoid irrelevant results such as "biographer" or "biogenesis") from Unpaywall were selected. Next, a list of biology-related journals from Wikipedia were acquired [7] as it provided the most comprehensive, up-to-date list found online which were also categorized based on specific topic (e.g.

biochemistry, nutrition, psychiatry, etc.) Papers contained in Unpaywall which were publicized across any of these journals were acquired. Then duplicates across both filtering methods were removed, allowing us to decrease the database size from 72GB to 24GB with a final set of 13.2 million journal articles.

Based on the same Wikipedia page, a list of 26 categories and their associated journals were extracted and compiled manually (refer to https://github.com/universvm/MediSia/blob/main/django_backend/medical_ir/data/journals/journals_categories.txt). This was done by removing any special characters and appending the list of journals relevant to each category, which is used for optimizing our retrieval model (Section 4). The list of categories is visible in Figure 1. An example of select corresponding journals for the category of "biochemistry" include journal publications such as the Annual Review of Biochemistry, Biocatalysis Biotransformation, and Journal of Structural Biology. Similarly, each journal article was tagged with a specific category based on its journal of publication; this was crucial in order to check that duplicates were avoided and users could have the option to filter search results by category if desired.

It is important to note that the journal names found in Unpaywall often lacked consistency; for example, due to the usage of acronyms. As a result, the same journal name could appear in multiple forms making the journal-based categorizing procedure problematic. To solve this issue, Natural Language Toolkit (NLTK) similarities were used to find the closest match in journal name. This procedure of tagging our 13M database took approximately seven days to complete. Although this procedure is not very accurate, it is consistent meaning it facilitates the query classification process.

agriculture	botany	medicine	ornithology
anatomy	dental	microbiology	pharmacy
biochemistry	ecology	molecular	psychiatry
bioengineering	entomology	mycology	virology
bioinformatics	forestry	neuroscience	zoology
biology	genetics	nursing	
biophysics	healthcare	nutrition	

Fig. 1: List of 26 categories relevant to the database.

B. Abstract Extraction

In order to optimize and improve the relevancy of the results, we additionally extracted the abstracts of each document to be used in addition to the document title for determining relevancy. This was accomplished by looping through all the papers and searching for an abstract from the following websites: bioRxiv, PubMed and crossref. These sites were chosen given their popularity, relevancy to scientific literature and large database sizes in order to cover as many papers as possible. If an abstract could not be retrieved from these sources, attempts to download it via a HTML page were made using the Selenium package to automate web browser interaction and Beautiful Soup for parsing. In order to cleanly extract abstracts on these websites typically characterized with noise from surrounding content (e.g. navigation bars, links), specific keywords were utilized to identify the start and end of abstract text. However, due to the range of website variation, consistency remained a challenge.

III. Indexing Approach

A. Preprocessing

Prior to creating the index, preprocessing steps were first taken with the input data which includes the title and abstract (if available) for each entry using text transformation. First, the cleaning function using regex removed any punctuation, HTML tags, special characters, double spaces as well as numbers found within words (e.g., hell3o becomes hello) and the minimum length of a word allowed was set to three to remove negligible words such as pronouns. Next included tokenization at non-letter characters, case folding, stopping followed by stemming using Porter stemmer from the Gensim library.

For stopping, we formulated our own tailored stopwords set combining scientific literature and medical database-specific terms for optimal preprocessing. Using the general English stopwords set from gensim as baseline, 190 additional medical stopwords were also added [9, 5]. The complete list of stopwords added can be found in *config.py* file and included terms such as "condition", "disorder", "drug", diagnosis", "pill", "specialist", "symptom" and "testing" which are more commonly found in medical literature as well as 10 miscellaneous terms commonly found in abstracts and reports, in general, which were "copyright", "facebook", "twitter", "email", "journal", "review", "volume", "date", "none", and "pdf." The stopwords list was expanded to ensure greater relevancy to the type of scientific literature-based data being used as opposed to a generic English stopwords set. These same preprocessing steps were utilized on the input query from the user to ensure matching.

B. Index Creation

Next, an inverted index was constructed to be used as a lookup table for streamlined retrieval in order to quickly find the relevant documents containing a specific term (i.e., token). This represents the main data structure typically used in SEs consisting of a term-index dictionary for each preprocessed term as keys and the corresponding postings list consisting of the list of documents that contain the term and a recording of its numerical position in a nested structure. This structure was critical to enabling TF-IDF vectorizer search by effectively combining the core database retrieved with SE capabilities, which will be discussed in the following section. Simultaneously, a document data store was created as a dictionary with its ID as the key and its URL, title, and abstract as its associated values, which was saved as JSOL file. Additionally, the vectorized form of all articles were saved to disk in effort to minimize time needed during the search.

C. Multi-Index Processing

Due to the large database size consisting of over 13M documents weighing at 24GB, it was imperative to find a strategy to limit the processing time needed for a search as opposed to inefficiently looping through every document in the database. The solution consisted of two parts: (1) creating multiple indexes based on category and (2) utilizing different types of search options geared towards the preferences and needs of the user. The latter will be discussed further in Section 4C. For the former, by creating an index per category, this allowed not only for multi-processing through multiple indexes at once (if multiple categories are relevant to the query) but also a targeted search through a specified index if the category of interest is already known, which helps speed up the search time significantly.

IV. Search Retrieval Process

In the following, we now shift to discussing the design for satisfying a user's request including, first accepting and classifying the query based on its relevant category (or categories), identifying and retrieving a set of relevant results as well as assisting the user in browsing and re-formulating their query if a follow-up search is desired for more targeted results.

A. Query Classification

When a free text query is provided by the user, it is first preprocessed using the steps detailed in Section 3A followed by the use of a classifier to determine the categories most relevant to the query. This is critical given that our initial screening revealed that simply searching for "coronavirus" across a whole index of 2M+ articles took nearly seven seconds whereas when this query was searched within individual sub-indexes such as virology (40K articles), biochemistry (195K articles) and medicine (1M articles), it took 0.81, 1.152, 3.41seconds respectively. This highlights the significant time save provided using a category-based index approach.

In order to first determine the relevant categories to be searched for a query, three types of classifiers and their performance were measured: Multinomial Naïve Bayes (NB), Bernoulli NB, and Stochastic Gradient Descent (SGD). Gaussian NB was also considered, however, its initial lack of performance and slow speed prompted its removal from the remaining testing phases. In order to determine the most effective classifier, all documents were obtained for each class and first divided into train, test and validation sets. The classifiers were trained using under sampling as well as random sampling, in other words, each class was made the same number of documents as the class with the lowest document numbers (i.e., under sampling) and at each epoch, the documents for classes which had more than the lowest numbers were switched randomly (i.e., random sampling). This ensured that at every epoch, the classifier was shown the same number of documents per class. Furthermore, at the end of every epoch, the documents of popular classes were switched to ensure more samples for that class were seen overall in order to avoid the idle case where the classifier simply selects the most frequent class to get the highest accuracy.

	Multinomial NB	**Bernoulli NB**	**SGD**
Accuracy	0.38	0.32	0.33
Top3	0.61	0.52	0.52
Top5	0.72	0.63	0.64
Precision	0.39	0.38	0.36
Recall	0.39	0.32	0.33
Time (s)	1.14	2.76	1.10

Fig. 2: Performance results of three classifiers used for query categorization; time refers to the time taken to classify 260K articles.

In terms of measuring performance, the following metrics were calculated: accuracy, top3, top5, precision,

recall and testing time (seconds) on a subgroup of 260K articles which are detailed in Figure 2. Based on these results, the Multinomial classifier was chosen as the default for its overall better performance.

B. Ranked Retrieval

MediSia's SE implements Term Frequency and Inverse Document Frequency (TF-IDF) with Cosine Similarity ranking as its main algorithm. The TF-IDF technique calculates the weight of each word in order to signify the importance of the word in the document and corpus while cosine similarity measures the cosine of the angle between two vectors projected in a multi-dimensional space which removes the effect of different length vectors (due to varying document lengths). In order to provide a ranked results list to the user, the following order of steps were taken:

1) Create a TF-IDF vectorizer object
2) Use fit-transform method to represent each document in the index of interest as a weighted TF-IDF vector and save into an object for use during query processing
3) Similarly, retrieve the TF-IDF vector representation of the query using the transform method
4) Compute the cosine similarity score for the select query against all the document vectors
5) Compile a results array ranking documents with respect to the query by score
6) Acquire the top 300 results to allow for follow-up search, if desired, and return the top 10 results to the user by pagination

Due to the large size of the dataset, the Gensim library was mainly used for optimization purposes; a generator was built on top in order to process documents larger than RAM, which was the case for our dataset. This enabled us to avoid having to load the whole corpus into memory. Furthermore, because our index was saved as a JSOL file, Python linecache was utilized in order to retrieve specific lines of the indexes quickly. The TF-IDF sparse matrix was saved as a Gensim Market Matrix and for calculating similarity, the similarity class (i.e. sparse matrix similarity class) was used. Figure 3 provides an illustrative overview of the system's search and retrieval design as we have detailed.

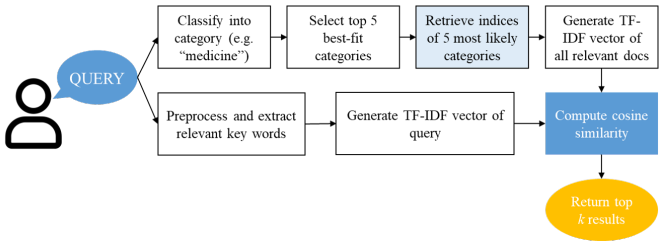

Fig. 3: Overview of search retrieval system.

C. Types of Search Functionalities

In order to provide a user-friendly experience which ensures both fast and accurate results are provided to the user, MediSia was uniquely designed to enable three types of search functionalities, which can be selected based on the trade-off preferences of the user. The three types of functionalities include: (1) Magic Search, (2) Topic Search, and (3) Deep Search. The unique aspects of each are detailed in the following and visualized in Figure 4.

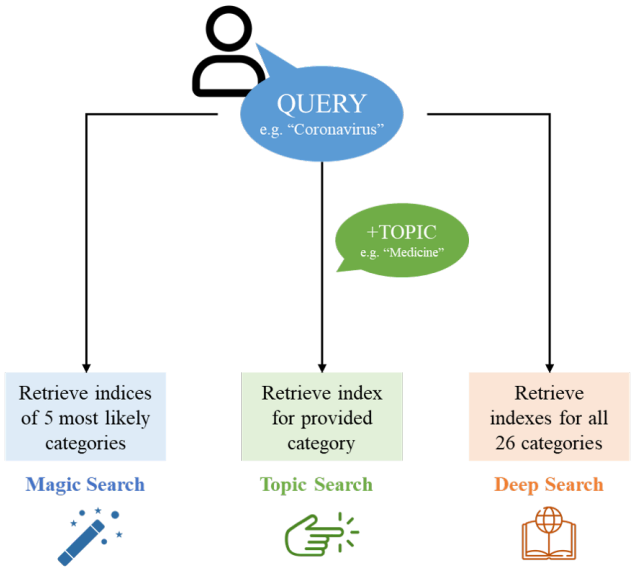

Fig. 4: Distinction across three search functionalities, based on Figure 3.

Magic Search (MS) serves as the default option for search functionality as it represents the option which best optimizes search accuracy with search time. It functions such that when a query is inputted into the system, a trained classifier (e.g. Naïve Bayes) is used to classify the query to determine the top five categories which are most relevant. This speeds performance such that only the indexes of these select five categories are searched, as opposed to the entire corpus. MS should be used when one wants to optimize the trade-off between

high accuracy of results and low search time.

Topic Search (TS) serves as the simplest and fastest option such that it is enabled when the user inputs a query as well as the specific topic he/she is interested in. This enables the system to look through only one select category (synonymous with "topic" which serves as the user-facing terminology) and its related index. Because the query does not need to be classified and only one index needs to be searched, this option provides the fastest response time.

Deep Search (DS) serves as the most comprehensive and in-depth search option such that it searches through the entire corpus, or all the indexes across all 26 categories. The total corpus consists of approximately 13M documents, therefore DS takes a longer processing time and is only recommended in cases where an exhaustive search is desired at the cost of time needed.

In addition to using our category-based indexing approach, another multi-processing strategy was implemented to further improve speed. In particular, the "medicine" category made up the largest segment of the corpus with over 5M documents out of a total of 13M documents, therefore encompassing over a third of our whole dataset. The other remaining 25 categories ranged between a few hundred thousand to a million documents per category therefore did not pose an issue. Given medicine's large, outlier size, it was split into six sub-indexes ("shards") of around 1M documents to allow for easier multi-processing and thereby quicker search time. A similar strategy was taken for DS functionality which initially took 67.8 seconds to search through for the query "coronavirus"; however, after dividing the entire corpus into 15 shards (less than 1M documents each), search time was reduced significantly to 24.2 seconds thereby supporting this approach.

V. User-friendly Interface

MediSia was designed with flexibility of user preferences in mind. Therefore, the search option was designed to include two parts: (1) the main search query and (2) an optional advanced search option, as seen in Figure 5. The main search query input space was designed to accept free-text query of any length. The advanced search option was enabled to give users the additional option to further filter their results in order to increase the relevancy of results.

In order to provide users direction and avoid any search errors, the "Search" button is disabled if no main

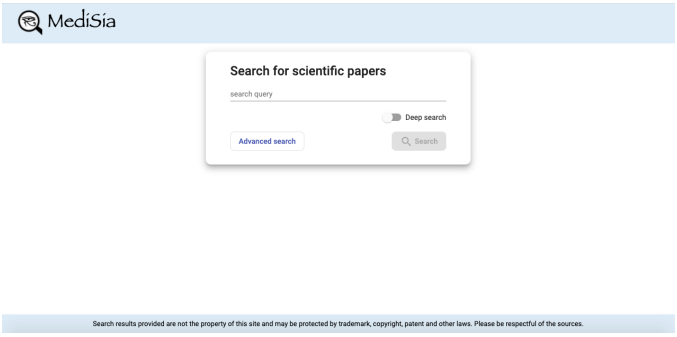

Fig. 5: Main search page view for query input.

query is provided. The search interface also includes a toggle button to allow users to activate (or deactivate) "Deep Search" functionality prior to proceeding with their search.

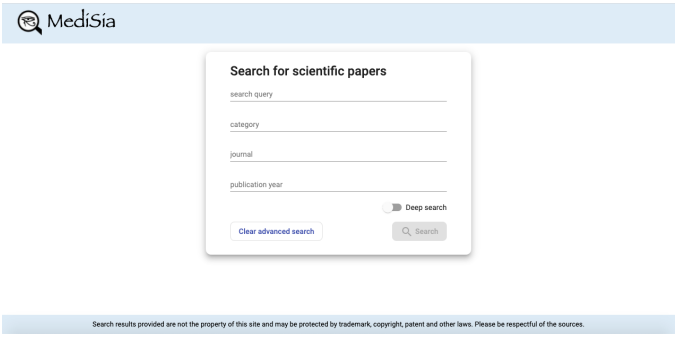

Fig. 6: Main search page view with advanced search option.

The advanced search option includes inputs for journal name, year of publication and category, shown in Figure 6. Entries for the publication year and category fields are validated prior to allowing the user to proceed with their search. The publication year field receives 4 digit years or ranges.

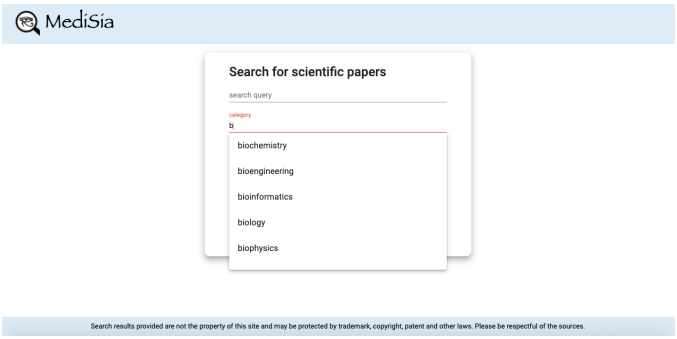

Fig. 7: Automatic dropdown of possible categories for selection provided as the user is typing.

The category field offers a drop-down of possible categories and filters the drop-down as the user begins

to type (Figure 7). The user must either type or select a valid category from the drop-down in order to enable the search button. This feature was designed to provide the user with flexibility to select which additional filters he/she wants to use and to help prevent spelling errors and invalid inputs being sent to the back-end.

A. *Viewing Results*

The results page displays both the results and the supplementary filter options. Using pagination, the top 10 relevant results are displayed initially on the first page. The user has the option to browse through additional pages, each with the next set of ranked 10 results.

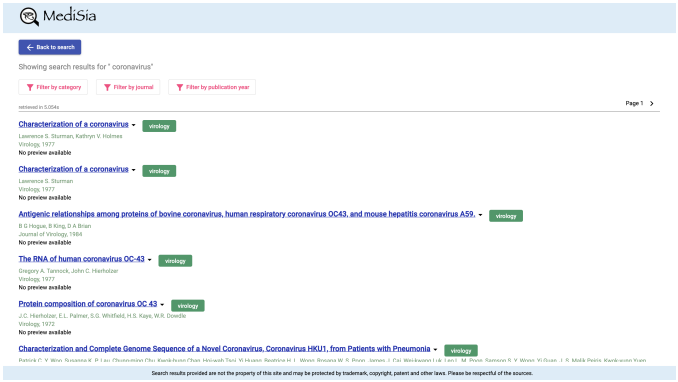

Fig. 8: Results page view with document links and clickable topic-based labels.

An example of a results page seen by the user can be seen in Figure 8. For each result, the title, authors, journal and publication year are displayed. Additionally, if an abstract has been found for this paper, a preview is shown. Furthermore, clicking on the title of the result will open a new tab which links to the paper, allowing the user to continue their search more easily, if desired. In an effort to add friendly visualizations of the document's topic to the user, each result is associated with a topic label which appears next to it and is color-coded by topic. The user can also choose to click the label which will then automatically filter results by the topic selected.

At the top of the results page, three buttons are shown which allow the user to further filter their retrieved results by category, journal or publication year. Clicking on one of these buttons opens a pop-up which displays a checklist of up to 10 options to filter by, based on the most relevant retrieved results. Figures 9, 10, and 11 provide visualization of these presented options.

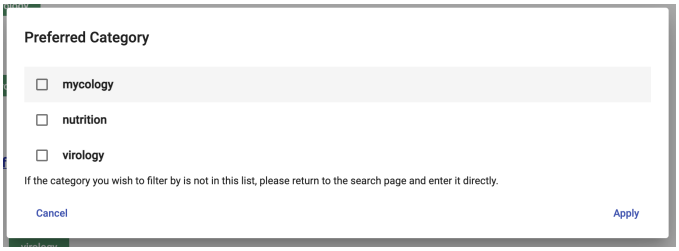

Fig. 9: Option to filter by topic category.

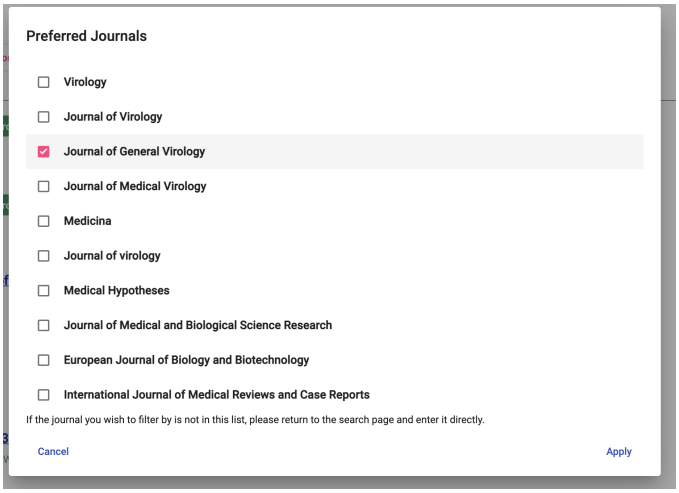

Fig. 10: Option to filter by journal name.

For example, if the user carried out an advanced search, any filters they originally applied will already be pre-checked on this list and can also be removed, if desired, by un-checking. If the user wants to enable additional filters, they simply have to check additional boxes and click the "Apply" button. This "follow-up" search is then sent to the back-end, which will send a new list of results according to the filters applied. The front-end will dynamically update itself to show these results once they are retrieved.

Fig. 11: Option to filter by publication year.

B. Performance Measure

For the purposes of measuring performance, the front-end logs certain metrics. Firstly, it tracks the time taken for a new search, both displaying this to the user (as seen in Figure 8) and logging it to allow us to measure the efficiency of the search function. The results interface also allows users to mark a retrieved result as irrelevant, giving us an additional metric by which to measure relevance. When a user hovers over the dropdown icon beside the title of a result, a small flag appears allowing users to mark this result as irrelevant, shown in Figure 12. When clicked, the result is stored as irrelevant for the search query it was returned for. If a user hovers over the dropdown for this result again, they will be shown the message "Marked as irrelevant; this result will be reviewed". Once the user begins a new search, the logged data for the previous search is saved, in order for us to improve the system.

Fig. 12: Option to mark a result as irrelevant appears by hovering on the dropdown icon.

To develop this front-end process, the web application TypeScript framework Angular was used, including the Angular Material UI component library for basic components to build the web page, functionality and select styling options. Additionally, Django was used as the back-end framework to receive the HTTP request and send relevant search results in JsonResponse back to the front-end. This allowed for seamless integration between the front-end and back-end framework. Redis was used as the cache solution to store generated results for corresponding follow-up search needs.

VI. Challenges Future Directions

The following discusses select challenges we faced during the development process as well as the solutions we implemented for these issues. In the case where there is potential for further development, ideas for improvement and scale are also presented. With time and resources available, these recommendations can be considered to design a public product for real-world use.

Some challenges that were faced originated from having a large dataset weighing at 25GB of 13M+ documents. This resulted in long processing times which limited the range of tasks we were able to execute. For example, abstract extraction for each document was extremely time-consuming; even with the use of multiprocessing, the extraction process took multiple weeks and timeout was eventually implemented. Additionally, there were some cases where abstracts were simply unavailable or extracted text was too noisy for proper analysis. Journal webpages are typically surrounded by a high degree of noisy content including navigation bars, links, advertisements, etc. making it a challenge to cleanly extract only the abstract across millions of documents. As a result, an effective approach to identify and extract relevant content requires prior knowledge and understanding about website-specific templates in order to hand-craft rules for extraction of the targeted content. This is a time consuming and difficult task and given the limited time of this project, we utilized a generic approach using Selenium (Section 2B) for this purpose as it does not require prior knowledge of website templates.

However, there is significant room for improvement with more time available through the use of text segmentation or targeted exploration of websites and HTML tags in order to enhance the quality of abstract text extracted. For example, an approach introduced in [6] combining HTML Document Object Model (DOM) analysis and Natural Language Processing (NLP) techniques for automated extractions of the text body and considering content subblocks to eliminate unwanted blocks from webpages (e.g. links or images) could be utilized. Similarly, this idea for improvement could be applied to our data sourcing as a whole. A tailored web crawler could be developed for both papers and abstracts in order to index a greater range of sources beyond only freely available papers and thus expand our coverage. For example, many scientific papers have restricted access without payment or journal subscription, therefore exploring tools which can provide in-depth crossref text and datamining service may provide additional results. This would also for continuous, live searching in order to keep our database up-to-date.

Another challenge relevant to having too large of a dataset relates to our choice of a search function. Prior to implementing our ranked retrieval approach (Section 4B), Boolean queries were initially explored (but excluded in our final product). With this search type, we encountered the “feast or famine” problem where either queries resulted in too many hits or too few (or none at all). These queries may be effective

for expert users who have a precise understanding of their needs and the collection at hand as well as the skill to formulate the query to produce a manageable list of results. However, for the majority of users and particularly web search, thousands of results are not helpful and a free text query is preferred given it better reflects human language over query language. MediSia is meant to be a user-friendly IR tool for both expert and non-expert users, therefore we decided to exclude this option and focus on optimizing our ranking algorithm which would provide tailored, more relevant results first, therefore the size of results would not be an issue and the user would not be overwhelmed.

To further enhance our retrieval system, another area for improvement would be the usage of word2vec embeddings in order to explore interesting properties between appearing terms and relevant document categories in which they are found. Vector representations would enable us to capture a range of information, meaning and associations such as the semantic and syntactic properties of input texts and further visualize similarities or dissimilarities between them. This could add value beyond our current cosine-similarity algorithm by allowing us to generate better distributed representations, introducing dependence of a term on other terms. Such embeddings would be obtained using methods such as Skip Gram or Common Bag of Words (CBOW) and algorithms such as word centroid similarity (WCS) could be implemented. This approach would represent each document as a centroid of its respective word vectors, encoding its "meaning" to some extent. Therefore, when a query is submitted, the centroid of its word vectors are computed and its cosine similarity to the centroids of matching documents can be used as a measure of relevance. The usage of our previous TF-IDF calculations would enable IDF re-weighted word centroid similarity (IWCS) in particular.

Our last recommendation relates to the final step of convincing the public to use MediSia for their medical knowledge needs in the future. For real-world implementation and scale, this system would need to be tested with researchers and potential users to determine the retrieval accuracy and provide empirical evidence of its utility. In order to test the effectiveness of our system as a large-scale Web application, a controlled experiment can be conducted. The evaluation methodology commonly adopted in the community is based on the Cranfield paradigm [3] using human judges to examine each of the returned documents to decide its level of relevance. This would enable us to evaluate our system's performance on a select effectiveness metric. Furthermore, to test the usability of our interface, User Experience (UX) testing techniques could also be conducted. For example, a heuristic evaluation by expert reviewers would allow us to compare our interface against accepted usability principles. Then to ensure a fluid, user-friendly experience, techniques such as usability lab testing with eye tracking and click tracking would provide us valuable information on the effectiveness of the linking structures on our website as well as the value of the supplemental features added to our system (i.e. category flagging, follow-up search options and filter types).

VII. Project Organization

A. Individual Contributions

The three main responsibilities of this project included: (1) back-end tasks (e.g., database indexing, query classification, retrieval model design, front-end with back-end connection), (2) front-end tasks (e.g., building the web interface) and (3) compiling findings and writing the final report. With four members in our group, these three main responsibilities were distributed across team members as his/her main priority. Given the variety of tasks required for (1) back-end-related tasks, this responsibility was split between Leo Castorina and Shuyu An, with Aimee Redbond responsible for (2) front-end tasks and Savina Kim responsible for (3) managing project progression and writing the complete report detailing our findings. The following provides further details regarding the individual tasks taken on by each team member.

Leonardo Castorina (s2090655)

- Creating a web crawler to download abstracts for all documents in the corpus
- Writing code for the process of vectorization into BOW and TF-IDF for all queries and documents
- Writing code enabling a searchable index and implementing a sparse cosine search function to determine document relevance for ranked retrieval
- Optimizing the search for speed and accuracy using multiprocessing and designing a "shards"-based technique
- Designing the query classifier with Magic Search functionality with alternative Topic Search and Deep Search
- Organizing tasks on Github Board for fluid project organization and other general management tasks related to questions to course instructors and submission (contact person)

Shuyu An (s2042303)

- Implementing a basic index generator, simple Boolean search the TF-IDF functions at initial stages of the project; however, updated search functions (i.e. Magic Search) were better optimized for use and replaced these functions in the final product
- Writing code to build the back-end using Django as web framework and combining the search modules with the back-end services
- Writing code to use Redis as cache service solution for follow-up search to obtain original results
- Responsible for back-end server tasks including connecting front-end with back-end using Django in order to receive HTTP requests from front-end and send back-end search results with JsonResponse to the front-end
- Developing web API interfaces and unifying related HTTP requests and response content with synchronous updates in cases of inconsistencies to ensure a fluid final system
- Carrying out the API interface testing for the front-end and back-end; integrating and debugging the final system

Aimee Redbond (s1713640)

- Designing the web interface and incorporating user-friendly features to enhance user experience and leverage our data to its full capability
- Writing code to create a search interface which takes user input to carry out searches, offering both simple and advanced search options to cater to different user types and validating user input prior to carrying out searches to prevent errors
- Processing data from user input to create and send a valid HTTP request, and later processing the back-end response in order to display results
- Writing code to create the dynamic results display, offering pages of retrieved results and filter options
- Writing code to allow users to apply filters to the retrieved results, using data from the back-end response to create lists of available filters and sending a follow-up search request when applied, with automatic display of new results once retrieved

Savina Kim (s2130984)

- Responsible for writing the full contents of the final report and designing all relevant features. This includes content for all eight sections detailing: (1) background knowledge on medical IR usage and sourcing gaps in current industry in order to relate these to MediSia's value-add; (2) database collection; (3) index creation and pre-processing details; (4) search retrieval model design; (5) front-end and user interface walk-through; (6) writing a summary of challenges, learnings and areas of future direction for potential improvement and scale; and (7) providing an overview of project management details throughout the coursework timeline
- Reviewing code related to index generation and search retrieval process
- Organizing tasks on Trello for fluid project organization and responsible for maintaining weekly meeting notes for progression tracking

B. Project Management

The project management component included weekly check-in meetings where each member would share their progress for the previous week and any new, additional tasks were delegated and concerns voiced. This allowed all members to be kept up-to-date with outstanding tasks and be distributed evenly to ensure no individual team member was overwhelmed; in these cases, other team members would voluntarily step-in to help alleviate the work needed. With regards to project organization, Trello was used for general task management and a GitHub board was used for specific code-related tasks. A private repository was used to share code among members and push/pull requests were enabled to ensure all code was reviewed by at least two or more members of the team prior to being published. Additionally, a group chat room was created via Microsoft Teams which allowed for live updates and immediate questions or concerns in between meetings to be solved quickly.

References

[1] Aysu Betin Can and N. Baykal. "MedicoPort: A medical search engine for all". In: *Computer methods and programs in biomedicine* 86 1 (2007), pp. 73–86.

[2] Yen-Yuan Chen et al. "Health Information Obtained From the Internet and Changes in Medical Decision Making: Questionnaire Development and Cross-Sectional Survey". In: *J Med Internet Res* 20.2 (Feb. 2018), e47.

[3] Cyril Cleverdon. *The cranfield tests on index language devices*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997.

[4] Radu Dragusin et al. "FindZebra: A search engine for rare diseases". In: *International Journal of Medical Informatics* 82.6 (June 2013), pp. 528–538.

[5] S. Gupta et al. "Research and applications: Induced lexico-syntactic patterns improve information extraction from online medical forums". In: *Journal of the American Medical Informatics Association : JAMIA* 21 5 (2014), pp. 902–9.

[6] Parag Mulendra Joshi and S. Liu. "Web document text and images extraction using DOM analysis and natural language processing". In: *DocEng '09*. 2009.

[7] *List of Biology Journals*. 2021. URL: https://en.wikipedia.org/wiki/List_of_biology_journals.

[8] Xiangming Mu, Hohyon Ryu, and Kun Lu. "Supporting Effective Health and Biomedical Information Retrieval and Navigation: A Novel Facet View Interface Evaluation". In: *J. of Biomedical Informatics* 44.4 (Aug. 2011), pp. 576–586.

[9] PubMed. *PubMed User Guide*. 2021. URL: https://pubmed.ncbi.nlm.nih.gov/help/#help-stopwords.

[10] *Unpaywall*. 2021. URL: https://unpaywall.org/.