Search Engine for CSE Website

by

LI, Kwan To
ktliac@connect.ust.hk
LUK, Sui Hei
shluk@connect.ust.hk
WONG, Yan Yuet
yywongaw@connect.ust.hk

Group 3

Advised by

Prof. LEE, Dik-Lun

Submitted in partial fulfillment
of the requirements for COMP 4321
in the
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
2020-2021

Date of submission: April 30, 2021

Contents

1 Introduction

Department of Computer Science and Engineering (HKUST) has been providing a wide range of support in educating students, researching and assisting growth in the IT industry. Its main page (www.cse.ust.hk) contains a variety of contents ranging from research to academics and admission to cater the interest of different users accessing the page. The goal of this project is to develop a web-based search engine to facilitate searching at the page which allows users to obtain related web contents by inputting queries. The search engine mainly includes a spider function, an indexer, a retrieval function and the web interface for query input.

This report will first introduce our overall system, database and user interface design. Then, the detail implementation of various parts of the search engine, such as vector space model, indexer, web interface will be further elaborated. We had also done different testings to ensure our program runs and works as expected and the procedures and results are listed. Following by that, is the overall review and feedback to the existing developed search engine and the report will end with detail procedures of installation and using the search engine.

2 Design

2.1 Overall System Design

The basic overall system design is shown in Figure 1 below and details for each part will be further elaborated.

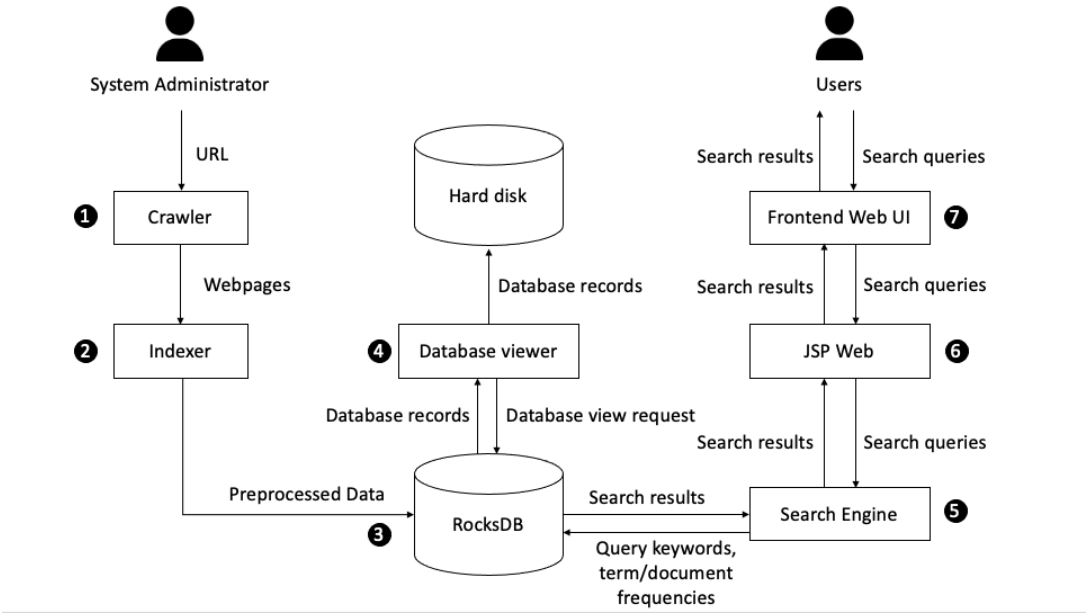


Figure 1: Overall system design

Crawler Given a starting URL from the system administrator, the crawler will recursively fetch the web pages from the given site into the local system using breath-first strategy which will be further elaborated below. For each fetched webpage, the crawler will further parse the HTML to extract all links to other URLs. After this, it will pass the fetched web pages to the indexer for indexing.

Indexer The indexer extracts keywords from the web pages passed by crawler and do stop word removal and stemming by utilizing Porter's algorithm and "stopwords.txt". Then, it will index document's URL and its basic details to Document ID Lookup Table and Forward Index Table respectively while details about keywords to Inverted Index Table at RocksDB.

RocksDB RocksDB is a database for storing key-value pairs data in byte array type which allows retrieval of content by its respective key. In the project, the indexer will be inputting indexed document's details to tables in the database. Database viewer will access relevant records in RocksDB by submitting database view request and receiving the output returned. Search engine will extract relevant information from the database for process by accessing the database with query keywords and term/document frequencies and receiving the search results returned from RocksDB.

Database viewer Database viewer is implemented in Phase 1 to read data from the forward index table stored at RocksDB and outputs the content to a plain-text file named "spider_result.txt" in the format as required.

Search Engine Search engine receives the search queries from user passes by JSP Web and compares the list of query terms found against different tables stored in RocksDB. For example, to support phrase search, we will check the positions of the query terms against the positions of the keywords in documents which is stored at the posting list of the Inverted Index table. Other functions including extended Boolean model and title search is also supported by the search engine in our current phrase. Here, term weighting formula is based on tfxidf/max(tf) and document similarity is based on cosine similarity measure. These formed the basis for ranking output order according to the vector space model after comparing against the existing the data in RocksDB. The search engine will further return these search results back to JSP Web for output display to user.

JSP Web JSP is used to directly access respective Java methods implemented in the search engine by putting Java inside HTML pages. It parses the search queries inputted from users via frontend Web UI and returns the results attained from the search engine back to frontend Web UI.

Frontend Web UI Frontend Web UI allows user to input search queries, passes the query to the search engine via JSP Web and displays the search results back to user in a formatted and clear manner. In this project, we support phrase search and the implementation of extended Boolean model such that user can have better specification to their query by utilizing double quotes, "AND" and "OR" when inputting their search queries. After receiving the search results from the search engine via JSP Web, the results will be displayed in a clear format with basic details of the documents, keywords, score and parent and child links which further elaboration on design will be mentioned below.

2.2 Database Design

We had three tables to support the system which are Forward Index, Document ID Lookup Table and Inverted Index. The Forward Index and Backward Index are the core of the search engine - the Forward Index maps document IDs to document record objects which contain details of the document while the Backward Index maps the keywords to posting lists which consist of the term frequencies of the words. Meanwhile, the Document ID Lookup Table is an auxiliary table that maps the URLs crawled to respective document IDs.

2.2.1 Forward Index

	Field	Type	Description
Key	Document ID	`Integer`	The document ID of the document to be looked up.
Value	Document Details	`DocumentRecord`	The document record details of the respective document.

The Forward Index table is used for storing the basic details of each document represented by their Document ID. The details to be stored include the title frequency table which stores the keywords in the title and respective frequencies to support title search, URL, last modification date, size of the page, content keywords and their respective frequencies, the child links, and the parent links.

Our design ensures the details of the document can be retrieved efficiently given the document ID. Here, we store the document record as a single object instead of using separate tables - one would be able to get all the details instantly in one single table access. This is especially important as the search engine is required to list out document details frequently during the search process. It would be too costly having to search through different tables looking up the results.

2.2.2 Inverted Index

	Field	Type	Description
Key	Keyword	`String`	The keyword to be looked up.
Value	Posting List	`Map<Integer, List<Integer>>`	The map that stores the term position of the respective keyword of documents.

The Inverted Index table stores the document IDs that contain the keyword and their respective positions in the document.

This is to cater the need for building a vector-space model later used for ranking the search results as well as phrase search in which details will be further explained below. In such a model, the similarity between a query and a document is to be calculated based on TF-IDF. That is, we have to support the efficient lookup of term frequency and document frequency given a set of query keywords which can be achieved simply by counting the number of positions a keyword has in the document.

When user input search queries, we can quickly lookup the inverted index table to get all the document IDs of the documents containing the keyword and also count the frequency of the keywords in the document. Instead of having to search through the values in the Forward Index

to find all the documents that contain the keywords specified in the query, the inverted index table speeds up the lookup process, thus giving better performance.

2.2.3 Document ID Lookup Table

	Field	Type	Description
Key	URL	`String`	The URL to be looked up.
Value	Document ID	`Integer`	The document ID of the document pointing to the respective URL.

The Document ID Lookup table stores the Document ID of a certain URL. By having an integer Document ID instead of a string URL representing each document, it reduces the storage and computational cost in storing document-related information in other tables and fosters easier and faster searching.

2.3 User Interface Design

We had implemented a Web-based user interface for getting user search queries and returning search results back to user. The following section will be a guide through our design.

2.3.1 Query Page

The home page of the Web UI is shown in Figure 2 which is displayed in a neat and clear format with the title "The Department of Computer Science & Engineering" indicating that this search engine is for content inside the main page of the department. Compounded with that, it is a simply text box allowing users to input their search queries.

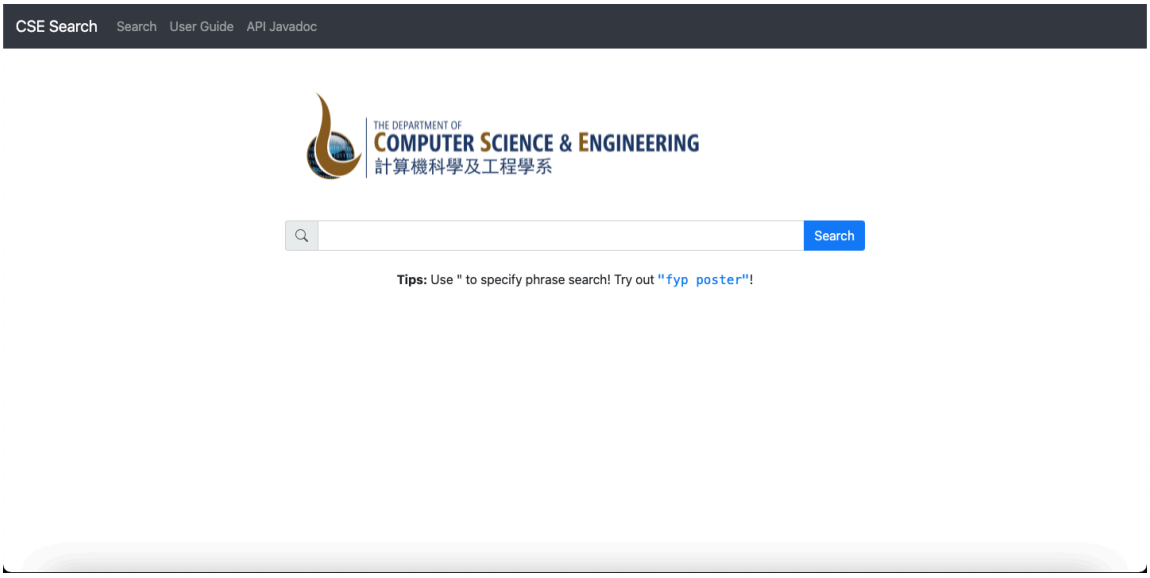


Figure 2: A screen capture of the home page

When the user is inputting a query to the text box, a drop-down list of query suggestions as shown in Figure 3 will be shown to the user, facilitating the user's input procedures. A simple "Search" button next to the query box is created such that the searching program can be triggered after user had finished inputting the search query and clicked the button.

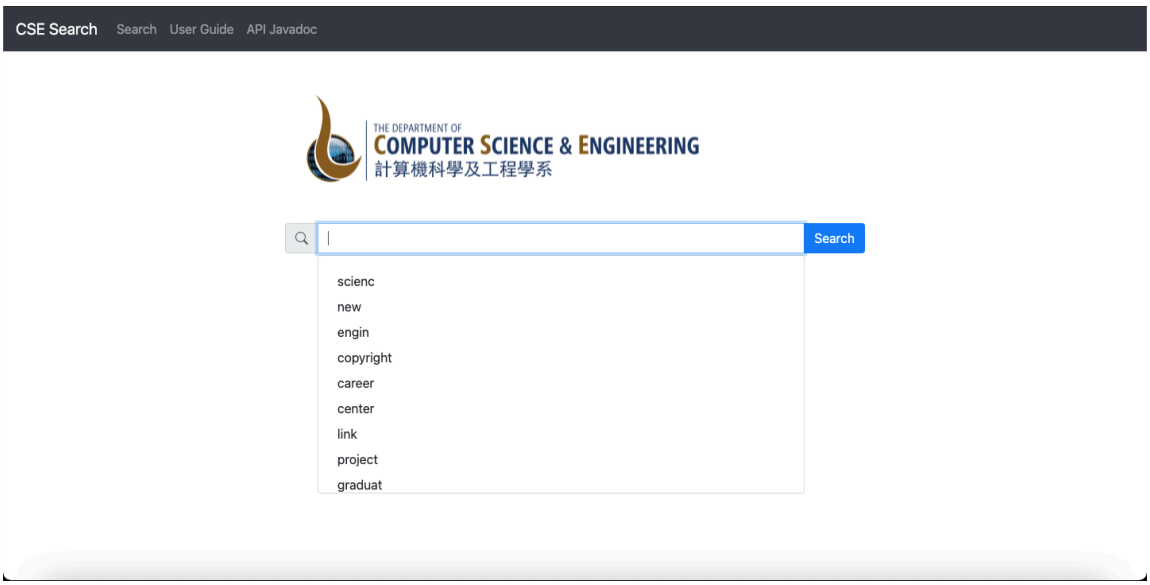


Figure 3: A screen capture of the home page showing query suggestions

2.3.2 Loading Page

As it may take some time to process complicated queries, we also implemented a loading page with a spinner as shown in Figure 4 to indicate that the program is currently running to retrieve related web pages.

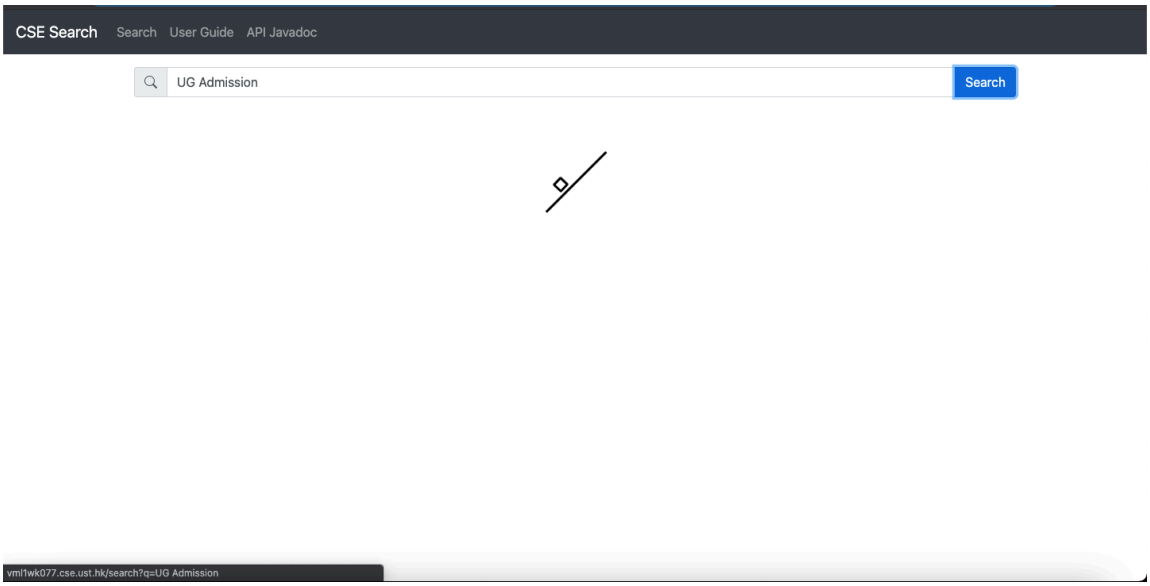


Figure 4: A screen capture of the loading page

2.3.3 Results Page

In displaying the results retrieved after the searching process, we target to display in a simple and clear format that provides all the information required for the project as well as being user-friendly. Listed below are two user interfaces for results page we had implemented to handle different search results.

If no relevant web pages are found, we will inform the user directly as shown in Figure 5. By providing a search bar above, user can quickly modify the current query to find relevant results.

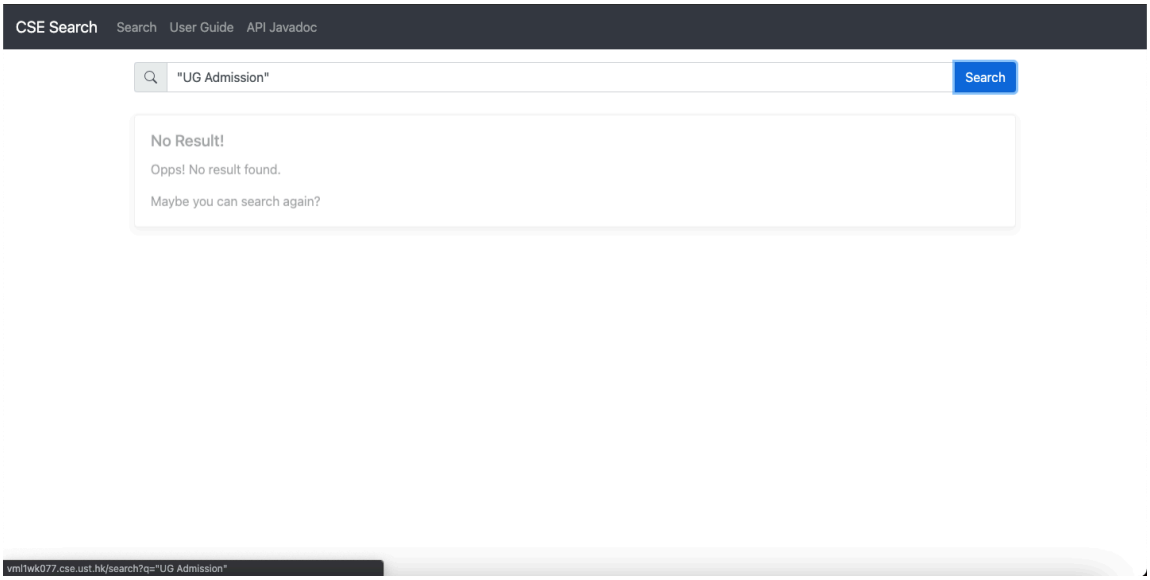


Figure 5: A screen capture of the result page showing no results

If some relevant web pages can be found, the results will be displayed in a format as shown in Figure 6. The display of each result web page is in accordance with the requirement specified for the project and ranked based on similarity score with highest displayed at the top. Here, the similarity score will first be shown in a coloured badge, along with other basic information including page title, url, last modification date, size of page, keywords and their respective frequencies and child links. For both url and child links, user can directly access the web page upon clicking the link, allowing user to quickly view the content of the related web pages.

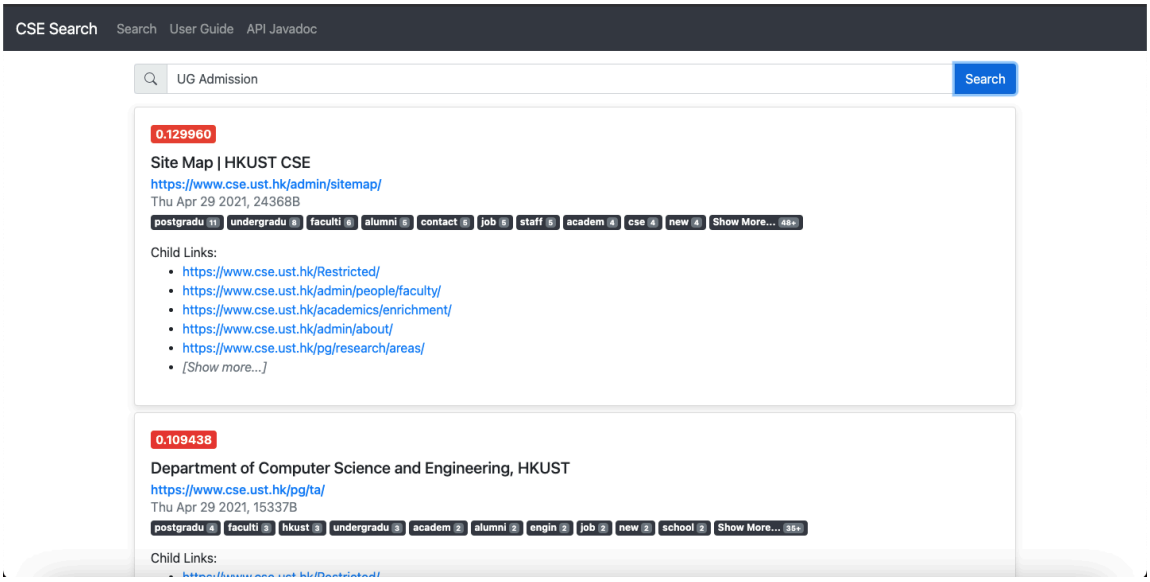


Figure 6: A screen capture of the result page list

3 Implementation

3.1 Search Engine

3.1.1 Vector Space Model

We adopted the vector space model for our search engine, as required in the project specification. Documents and queries are represented as vectors for performing vector operations in order to compute statistics for ranking.

To implement the model, we represent each vector as a map — keywords string as the key and the term weight as the value. Hashmaps are used to allow efficient retrieval and update of word weights. Utilities functions are developed to perform vector operations such as addition, element-wise product, etc.

3.1.2 Term Weighting

To represent the document as a vector, we used the normalized TF-IDF weight as specified in the project requirements. We get both the term frequency and documents frequency from the inverted index. The frequencies are then normalized and computed as the final term weight using the formula: $w_i = \frac{tf_i}{\max tf_i} \cdot \log_2 \frac{N}{df_i}$. Note that we have tried to batch the lookups to reduce duplicated lookups in the inverted index.

3.1.3 Document Ranking

To rank the documents, we are using a document-query similarity score. The similarity metric that we used is cosine similarity, as specified in the project requirement. For each query, we compute the similarity between the query and each of the document accordingly. The cosine similarity is calculate as the dot product between the query vector and the document vector, normalized by the L2 norms of both the query vector and the document vector.

One implementation detail to note is that we only consider documents containing at least one of the keywords. Documents that do not contain the keyword would have the dot product equals to zero, leading to 0 similarity as a result.

Title match is supported by utilizing the title frequency table which stores the keywords in the title and respective frequencies stored in Forward Index Table. When user inputs search queries, instead of only considering the term frequency within document content, matches in document title will also be considered. In this project, the program will return the weighted sum of frequency which is achieved by a configurable constant currently set to be 1.2, meaning that any match in title will count 1.2 times as compared to usual match in content.

3.1.4 Phrase Search

User can specify phrases in their search queries by using double quotes. Phrase search is supported mainly by the Inverted Index table. For each of the keywords in the phrase, the program will first get all the document IDs that contain the keyword. For example, if the phrase search query is “Computer Engineering” , the program will get a posting list of document IDs and keyword positions that contain the keyword “Computer” and another list that contain the keyword “Engineering”simply by searching at the Inverted Index table. Upon having these lists, we will do intersection to get documents that contain all the keywords in the phrase. Finally, by comparing the position of each keyword in the posting list with the order as inputted by

user, we can check if the phrase truly exists in the document. Figure 7 below shows an example to illustrate the implementation of phrase search.

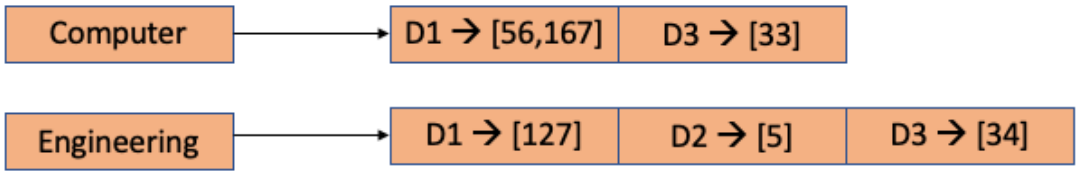


Figure 7: Phrase Search Example

Above is a make-up sample data from the Inverted Index table which each keyword points to all document IDs that contain the keyword and respective locations of the keyword. If the user input "Computer Engineering" as query and specified with double quote to indicate phrase search, we will first get all document IDs that include any of the keywords, which in this case all D1, D2 and D3 will be chosen. After doing intersection as mentioned above, documents left will be D1 and D3. Finally, with the step of order checking, we can see that only D3 has the two terms in order as specified by the user, so D3 is the only document that satisfies user's phrase search in this example.

3.1.5 Bonus Feature: Extended Boolean Model

We allow specifying multiple queries connected with boolean operators (AND, OR). We have implemented the extended boolean model introduced in the lecture to support those queries. To get the similarity score, we would take the square root of the average squared similarity of the sub-queries. In our implementation, the parser will detect the "and" and "or" keywords in the query and parse the query into multiple sub-queries accordingly. When executing the query, the sub-queries will be executed and the similarity score is calculated with an implementation of the aforementioned formula.

3.2 Indexer

3.2.1 Crawler

Crawler will fetch webpages that previously had not been fetched or webpages that had been modified after the last fetch. These can be found by checking if the URL of the webpage exists in the Document ID Lookup table and looking at the last modified date stored in the Forward Index table respectively. With the given starting URL, the crawler will recursively fetch webpages into the local system. For each webpage fetched, it will extract all hyperlinks so that it can continue the recursive fetching process. In this project, we had mainly utilized jsoup library and java.net package to complete the implementation. The crawler uses breadth-first strategy which the crawler will start with the root URL and fetch all of the URL in the present depth before moving on to fetch links pointed from the current fetched URL. In other words, the crawler fetches the webpages layer by layer. As compared to other search strategy such as depth-first search which fetches all URL from the current URL before moving to the next webpage in the same level, breadth-first strategy fosters a more even output of contents from different sections.

3.2.2 Stopword Removal and Stemming

To improve the performance of the search engine, stopword removal and stemming are implemented. Stopword are frequently-used words that do no carry much meanings, such as "a", "the". These words are assumed not to carry any important information and are ignored and removed during page processing in order to save storage space. In the implementation, we utilized the file "stopwords.txt" provided which contains 1298 stopwords and removed these stopwords from the document crawled to prevent indexing unimportant terms.

As for stemming, it is the process of removing affix and successor with the assumption the words having the same stem usually have similar meaning. It improves the recall of the search by getting documents that are relevant to user's queries while not having the exact terms in the query. We utilized the Porter algorithm implemented in the program "porter.java" provided which is a simple algorithm having more than 40 years of history for stemming English language words. It includes context-free and context-sensitive rules to either remove the suffix or transform it to other forms for enhancing the performance of the search engine.

3.2.3 Database Indexing

To update the index after crawling, we need to first lookup the URL in the Document ID Lookup Table and assign a new Document ID to the URL if it does not exist. Then all the details of the document will be further updated to the Forward Index along with the assigned Document ID. To update this record to the Inverted Index, we would lookup each keyword specified in the document basic details and further update the posting list if needed. For a new keyword, we would insert a new key-value pair by having the keyword pointing to DocID and the frequency of the word in the document. If the keyword is found, we would simply add the DocID and frequency to the posting list.

3.2.4 Bonus Feature: Query Suggestion

We have implemented an index that will give the top 20 most frequent words with a certain prefix. This is used for suggesting the full query word when the user is typing the query. For example, when a user type "rep", the search engine may suggest words like "repeat", "report", etc.

However, such an index is pretty difficult to build as RocksDB does not support suffix truncation by default. We have implemented the Trie data structure to help building the index efficiently. As we are updating the inverted index in batches, the posting lists for the words are stored in memory. We may used the posting list to efficiently get the document frequency and insert it into the Trie.

To obtain the result for certain prefix, we use a DFS-like process to merge the child nodes getting the most frequent word list. The list of words starts merging from the bottom and only the top 20 words would be kept inside each node. The final list would then be stored in a RocksDB index for fast retrieval during search time.

3.3 Web Interface

A video demonstration of the web interface is provided in https://youtu.be/mjxxL2PuKzs.

3.3.1 Java Webpages

To support old browsers, a simple webpage without fancy UI is developed. The web server is directly bundled with the search engine — it uses JSP to directly access the Java methods implemented in the search engine. The page parses the query parameter of the request and get the result from the search engine by calling the respective method. Such as simple UI also allows us to test our application quickly without worrying problems from the complicated UI logic.

3.3.2 Bonus Feature: JSON API

To bridge between the backend search engine and the frontend UI, a JSON API is developed. We decided to opt for client-side rendering where the browser dynamically build the HTML DOM according to the data received from the server. We use Jackson, an industrial standard JSON library, to parse the search result Java objects into JSON String for serving in the JSP pages.

The popular JSON format is compatible to a numerous JavaScript framework which allows the frontend to parse the data easily. The API also allows other external applications to be developed to suit the client's needs.

3.3.3 Bonus Feature: SPA with AJAX

We have choosen React.js for building our user interface. With the React.js framework, html components are loaded dynamically according to the application state and the request endpoint. When browsing through pages, there is no necessity reloading the whole page giving a fast and reactive experience to the user. For loading the content, such as the query result, an AJAX request call will be send to the JSON API for getting the data. Then, it will be parsed and displayed to the user accordingly.

3.3.4 Bonus Feature: Responsive Design

The user interface is mainly developed using the Bootstrap framework. Components of the frameworks are used for a consistent look-and-feel throughout the user interface. The components are responsive to different screen sizes, enabling a comfortable across devices. The framework make uses of CSS media queries to apply different CSS styles depending on screen size. We have implemented our UI following the framework guidelines accordingly.

4 Testing

4.1 Unit Testing

We have implemented a few unit tests with JUnit to ensure the core functionality of the search engine works as expected. For example, we have tested our vector space model implementation by explicitly testing core functions like vector addition, dot product, and cosine similarity. The set of unit test will be run each time after we incorporates new features such that all of the old function would work as expected.

4.2 Integration Testing

To ensure our system works as a whole, we have designed a testcase which goes through the every component of the system. The major steps include:

- Crawl & index pages from the CSE website with specified no of max pages.
- Repeat the above to ensure the index can be updated correctly.
- Print the index files to a file.
- Deploy the search engine using the indexed pages.
- Perform a few search using the user interface.

We conducted the test periodically to ensure there is minimal error when all of the components are integrated as a whole.

4.3 User Acceptance Test

We have invited some of our fellow classmates in the CSE department to try out our search engine. They are asked to input some queries and mark whether the first 5 results are relevant or not. They are also asked to give feedback on the user interface design and the overall user experience.

5 Discussion

5.1 Search Result Evaluation

To evaluate the performance of our web-based search engine, we had come up with 3 test queries and explicitly evaluated top 5 results returned to calculate the precision.

5.1.1 1st Test

The first test we will use "goal" as our input query, our target is to know more about the purpose and aim of CSE Department. The top 5 results are as shown below.

Rank	Similarity Score	Document
1	0.201209	Distributed Algorithms for Computing Statistical Information on Massive Data | HKUST CSE
2	0.140688	Secure and Private Data Management | HKUST CSE
3	0.137282	Modeling and Verification of System of Systems | HKUST CSE
4	0.121642	Survey on Personalized Information Retrieval | HKUST CSE
5	0.118272	Constructing Knowledge Representation From Lecture Videos Through Multimodal Analysis | HKUST CSE

We found none of the top-5 results to be relevant. The top-5 precision is $\frac{0}{5} = 0$. As we are attempting to find the purpose and aim of CSE Department, however, the displayed results are mainly about the goals for different research projects, forums and seminars. Thus, they are considered as irrelevant.

5.1.2 2nd Test

After the first test, we understand that the query terms might not be good enough to achieve satisfactory results. Thus, the second test we will use "mission and vision" as the input query, with the same target to know more about the purpose and aim of CSE Department. The top 5 results are as shown below.

Rank	Similarity Score	Document
1	0.127680	Site Map | HKUST CSE
2	0.107519	Department of Computer Science and Engineering, HKUST
3	0.037815	Site Search | HKUST CSE
4	0.022466	Information for Employers & Industry Partners | HKUST CSE
5	0.021899	三值光学计算机的存储器结构及其对电子计算机结构的改进 | HKUST CSE

We would consider #1 and #3 as relevant. The top-5 precision is $\frac{2}{5} = 0.4$. They are considered as relevant as both of them provide a quick pathway to link to the mission and vision of the CSE Department. As for other web pages, they are more correlated to other areas which is different from my target.

5.1.3 3rd Test

In the third test, we would like to know more about machine learning and used "machine learning" as our query. Here, we used phrase search attempting to increase the chances of getting relevant results. Below is the top 5 results outputted.

Rank	Similarity Score	Document
1	0.622298	A System for Large Scale Machine Learning | HKUST CSE
2	0.621187	The Power and Limits of Machine Learning Models | HKUST CSE
3	0.503308	DMTK: Making Very Large Scale Machine Learning Possible | HKUST CSE
4	0.498492	Lifelong Machine Learning Literature Survey | HKUST CSE
5	0.443052	Supervisionless Machine Learning with World Knowledge | HKUST CSE

We would consider all of the top-5 results to be relevant. The top-5 precision is $= \frac{5}{5} = 1$. They are considered as relevant as all of them are related to machine learning and are providing information and analysis on this topic.

5.2 Limitations

5.2.1 Capture Text Ordering

In ranking the web pages, our program mainly takes frequencies of keyword match into consideration but fail to capture text ordering if phrase search is not specified, meaning that the order of each keyword in a query is ignored. For example, queries of "machine learning" and "learning machine" are going to return the same output results while user might be looking for different things. For "machine learning", user might intend to know about different machine

learning algorithms and analysis on related topics. As for "learning machine", user might be looking for details and information related to machine, instead of specifically machine learning. Thus, we can see that the order of keywords can imply different targets during the search and can better boost performance of the search engine if text ordering is captured.

5.2.2 Language Understanding

From the first and second search tests mentioned above, we can see that while "goals" and "mission and vision" are highly similar, the results returned can be very different, resulting in different precision. For both queries, we intend to know more about CSE Department's purpose and work but without exact match in keyword "mission and vision", we cannot retrieve any relevant results. Thus, we can see that the current search engine does not support synonym but requires exact match in keyword to be counted as relevant. Natural Language Processing can be incorporated to not only consider the frequency of keywords during search, but also the meaning of each terms. This can better improve the performance of our search engine by outputting more relevant results to user.

5.3 Potential Additional Features

5.3.1 Relevance Feedback

Relevance feedback is the reformulation of query or modification of document after receiving user's feedback, with an aim to better return outputs users desired. It is achievable explicitly by creating a few buttons or icons aside each search result to allow user to indicate if one finds that relevant. It can also be done implicitly by observing user's behaviour such as clicks to identify what user truly desires. Having these information, we can modify user's query by amending the weighting of the terms more similar to those in relevant documents and different to those identified to be irrelevant. By doing this, the search engine will take in users' feedback and make adjustments based on that, which can highly enhance the performance.

5.3.2 PageRank

In our current search engine, only term-based ranking is used in ordering the results. However, a web page with lots of relevant keywords do not necessarily mean it's relevant. We should take in other factors to ensure the quality of top-presented results. One of them is link in which in PageRank, the importance and quality of a page is derived by the number of inbound links, indicating that the page is usually of higher importance when more pages had cited that in own page. Our current implementation has listed the child and parents links of each web page. We can enhance the performance by pushing it further to include the number of parent links in evaluating the score of a web page. By that, we can better ensure top results are of high vote of confidence by other web page publishers and are likely to be trustworthy.

6 Project Management

6.1 Version Control

We have used Git for version control in the project. All of our code is stored in two private repositories in GitHub, one for the main search engine, and one for our user interface. We have setup a CI pipeline in GitHub. The CI will build and run all the unit tests to ensure there is no major errors when the changes are integrated in the main branch.

6.2 Division of Labour

Table 1 shows the task distribution within our group.

Task	LI, Kwan To	LUK, Sui Hei	WONG, Yan Yuet
Vector Space Model			
Extended Boolean Model			
Title Weighting			
Phrase Search			
RocksDB Interface			
Crawler & Indexer			
Stopword Removal & Stemming			
Search Suggestion			
UI Design			
Unit Testing			
Integration Testing			
Performance Evaluation			
Project Report			
Installation Guide			
User Guide			
Javadoc			

(Color Code: Accountable Responsible Consulted)

Table 1: Division of Labour

Although the number of tasks completed differ among different team member, we all agreed that each of us has put considerable amount of effort for this project. No one should be penalized due to the fact that he/she did less effort relative to other team members. Therefore, we would like to declare our contribution percentage as the following:

- LI, Kwan To: 33.33%
- LUK, Sui Hei: 33.33%
- WONG, Yan Yuet: 33.33%

6.3 Documentation

To increase reusability of our system, we have included Javadoc for some of the public classes in our project. With clear documentation, alternative search engines can be developed based on our implementation to improve their search results. Meanwhile, the clear documentation enables our team to work efficiently without spending a lot of time understanding other's codebase.

7 Conclusion

To conclude, we had successfully developed a web-based search engine that facilitates searching at the main page of Department of Computer Science and Engineering (HKUST). The search engine mainly includes a crawler that recursively fetch documents into local system, an indexer that pre-processes the documents and pass to respective tables in RocksDB, a retrieval function that rank the documents based on user query according to the vector space model and a web interface for query input and results display. Besides the basic requirement for the project, we had implemented some extra bonus features including the extended boolean model in search

engine, query suggestion in indexer and JSON API, SPA with AJAX and responsive design in web interface, aiming to enhance the performance and user-friendliness of our search engine. We had also identified limitations and potential additional features to our current system which we believe through further development and enhancement of these items, the performance of our search engine can be further improved.

A User Manual

A.1 Demo Deployment

An instance of the search engine is deployed at http://vml1wk077.cse.ust.hk/ for demo.

A.2 Installation Guide

A.2.1 Installation Environment

We recommend running the application in an UNIX-like environment. It is not required but some of the scripts may not be able to run outside UNIX-like environment.

The project is well-tested in the following environment:

- CentOS 7 (Cloud VM environment)
- Arch Linux
- MacOS Big Sur

Although theoretically this project should have no problem running in Windows, it is not tested and not guaranteed if it would work.

A.2.2 Types of Installation

There are several setups for deploying the search engine stack. Choose the one that suits your needs and constraints.

- Hassle-free Deployment with Docker (HDD) (Recommended)
- Tomcat + Static Web Server
- Tomcat only

A.2.3 Hassle-free Deployment with Docker (HDD)

The setup is preferable if you have Docker installed in your machine. You may deploy the full search engine stack only with a few commands.

Pre-requisite

Java

Crawling needs to be done before launching the container. Therefore, it is necessary for the host to have JDK installed. JDK 8 is recommended because the project is developed with Java 8.

Please make sure the java binary directory is in *$PATH* and *$JAVA_HOME* is set to the Java installation directory, e.g. */usr/bin/java*.

Docker Installation

Please refer to the Docker website for installation of Docker in different distributions.

Installing Docker in CentOS:

1. Uninstall old versions of docker

```
sudo yum remove docker \
          docker-client \
          docker-client-latest \
          docker-common \
          docker-latest \
          docker-latest-logrotate \
          docker-logrotate \
          docker-engine
```

2. Set up repository

```
sudo yum install yum-utils
sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
```

3. Install docker engine

```
sudo yum install docker-ce docker-ce-cli containerd.io
```

4. Start Docker

```
sudo systemctl start docker
```

5. Verify that docker is installed correctly by running the `hello-world` image

```
sudo docker run hello-world
```

For more information, please refer to docker's installation guide.

Docker Compose Installation

Please refer to the Docker website for installation of Docker Compose in different distributions. Installing docker compose in CentOS:

1. Ensure required dependency packages are installed

```
sudo yum install py-pip \
                 python3-dev \
                 libffi-dev \
                 openssl-dev \
                 gcc \
                 libc-dev \
                 rust \
                 cargo \
                 make
```

2. Curl a stable release of docker compose

```
sudo curl -L \
"https://github.com/docker/compose/releases/download/1.29.1/docker-compose-$(uname -s)-$(uname -m)" \
-o /usr/local/bin/docker-compose
```

3. Add executable permission

```
9ptsudo chmod +x /usr/local/bin/docker-compose
```

4. Create symbolic link

```
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

For more information, please refer to the installation guide of docker compose.

Starting the stack

It is recommended to use docker compose for starting up the whole search engine stack instead of installing it separately. It is done in the backend directory `comp4321-proj` where `docker-compose.yml` resides.

By default, it assumes the RocksDB is stored in `$PWD/db`. You may want to change the mount path or use docker volume instead by modifying `docker-compose.yml`.

It is required for the web to be crawled for at least once. Please refer to Crawling and Prerequisites section below.

```
docker-compose build
docker-compose up -d
```

You may want to change the listen port in `docker-compose.yml`. The default port is `3000`.

A.2.4 Tomcat + Static Web Server

Pre-requisite

Java

The project runs with Java 8. It is recommended to have JDK 8 installed.

Gradle is used for dependency management for this project. The gradle wrapper script will be able to detect your Java installation. Make sure the java binary directory is in `$PATH` and `$JAVA_HOME` is set to the Java installation directory, e.g. `/usr/bin/java`.

Tomcat

The search engine backend APIs are served with `Apache Tomcat` server. The recommended version is `10.0.5`.

Download and extract the package to any location you prefer. Alternatively, you may install Tomcat through your system's package manager. You may want to set the environmental variable `CATALINA_HOME` variable to the installation location `apache-tomcat-10.0.5`.

Node.js and Yarn

The project is created with `create-react-app`. To setup the dependencies for building the app, the yarn package manager needs to be set up.

To install nodejs (and npm), it can be done with package manager.

Debian and Ubuntu:

```
sudo apt update
sudo apt install npm
```

CentOS 7:

```
curl -L https://rpm.nodesource.com/setup_10.x | sudo bash -
sudo yum install nodejs
```

MacOS:

```
brew install yarn
```

To install `yarn`, it can with installed with npm

```
sudo npm install --global yarn
```

Please refer to the yarn website for alternative setup instructions.

Static Web Server

To serve the UI website, you will need a static web server to serve the files. Moreover, it should be able to proxy requests to support API calls to the backend JSP web server. We recommend using `nginx` for this purpose.

Building the Backend

The commands and relative paths listed in this section `Building the Backend` are executed in/relative to directory `comp4321-proj/`. Please change directory into the directory `comp4321-proj/` to execute the commands.

Dependency Management

Dependency in backend directoory `comp4321-proj` is handled by Gradle.

Building the CLI for backend

For simplicity, you can build a jar file with all libraries packaged in a single jar. Run the shadowJar Gradle task for building a fat jar.

```
./gradlew shadowJar
```

The compiled jar file will be located at `build/libs/COMP4321Project-1.0-all.jar`.

Building the Web Application for backend

Web application WAR file can be built for deployment with Tomcat server through `war` Gradle task.

```
./gradlew war
```

The compiled war file will be located at `build/libs/COMP4321Project-1.0.war`.

Building the Frontend

The commands and relative paths listed in this section `Building the Frontend` are executed in/relative to directory `comp4321-proj-ui/`.
Please change directory into the directory `comp4321-proj-ui/` to execute the commands listed below.

Dependency Management

Dependency in frontend directory `comp4321-proj-ui` is handled with yarn package manager. All the dependencies should first be installed with the following command.

```
yarn install
```

Building the frontend UI

To build the static web pages, you may use the `build` script in `react-create-app`. The resulting pages will be stored in the `./build` folder.

```
yarn build
```

Deploying the UI

The commands and relative paths listed in this section `Deploying the UI` are executed in/relative to directory `comp4321-proj-ui/`. Please change directory into the directory `comp4321-proj-ui/` to execute the commands listed below.

Setting up the Static Web Server You may setup any of the static web server of your preference. Copy all the build files to the web server directory for serving. The below command assumes nginx server is employed and files in `/usr/share/nginx/html` are served as the root.

```
cp -r ./build/. /usr/share/nginx/html
```

The setup can be verified by loading the home page.

Linking the Backend API

For the actual search functionality, you need to setup proxy to point all `/api` requests to the backend JSP server. For example, if you are using nginx, you may want to setup like the config below (assume backend is deployed at port `3001`) .

```
server {
    location /api {
        proxy_pass  http://localhost:3001;
    }
}
```

Deploying the Backend Search Engine

The commands and relative paths listed in this section `Deploying the Backend Search Engine` are executed in/relative to directory `comp4321-proj/`. Please change directory into the directory `comp4321-proj/` to execute the commands.

The search engine web application is packaged in a single WAR file. There are several possible ways to deploy it.

Crawling and Pre-requisites

Before deploying the web application, it is required for the pages to be crawled and indexed into RocksDB for at least once. It can be done with:

1. convenient script `phase1.sh`

   ```
   ./phase1.sh --crawl --num-docs=20000
   ```

   Options available for the script can be shown by

   ```
   ./phase1.sh --help
   ```

2. shadowJar built in a Deploying the Backend Search Engine section before

   ```
   java -jar build/libs/COMP4321Project-1.0-all.jar --crawl --num-docs=20000
   ```

   Again, available arguments can be list by

   ```
   java -jar build/libs/COMP4321Project-1.0-all.jar --help
   ```

Environmental variable `SE_DB_BASE_PATH` needs to be set to the RocksDB storage folder. By default, the RocksDB is stored in `$PWD/db`.

Method 1: Convenient Script

To your convenience, we provide a script that do all the things for you. The script will download and extract Tomcat automatically. Then, the web application will be built (with `war` Gradle task) and deployed to Tomcat.

To start the server, run the following:

```
./phase2.sh startup
```

By default, the application is deployed at port `8080`. You may visit http://localhost:8080 to see it in action.

To stop the server, run the following:

```
./phase2.sh shutdown
```

The port can be changed by modifying `tomcat/conf/server.xml` AFTER running both `startup` and `shutdown` of `phase2.sh`. By changing `port="8080"` in the following block to `port="3001"` or other desired port, the Tomcat server will serve at port `3001` or other port when it is started.

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443"/>
```

Method 2: Manual Deploying

The WAR file is located at `build/libs/COMP4321Project-1.0.war` after running the `war` Gradle task. You may follow the instruction of your web server distribution for deploying a WAR application.

For Tomcat, copy the generated WAR file to `$CATALINA_HOME/webapps/` to deploy the application. You may refer to the official documentation for details.

A.3 Using the Search

This part will be a detailed guide of using the web-based search engine we had developed.

A.3.1 Inputting Query

In inputting a search query, user can simply type in the keywords in the text box as shown in Figure 8. There will also be a drop down list of suggestions based on the prefix a user had inputted. If user finds any of the suggestions useful and related, user can simply click on the suggestion, then the input query will automatically selected to be the selected suggestion. After typing in a search query, user can select the Search button next to the text box to start the triggering the searching process.

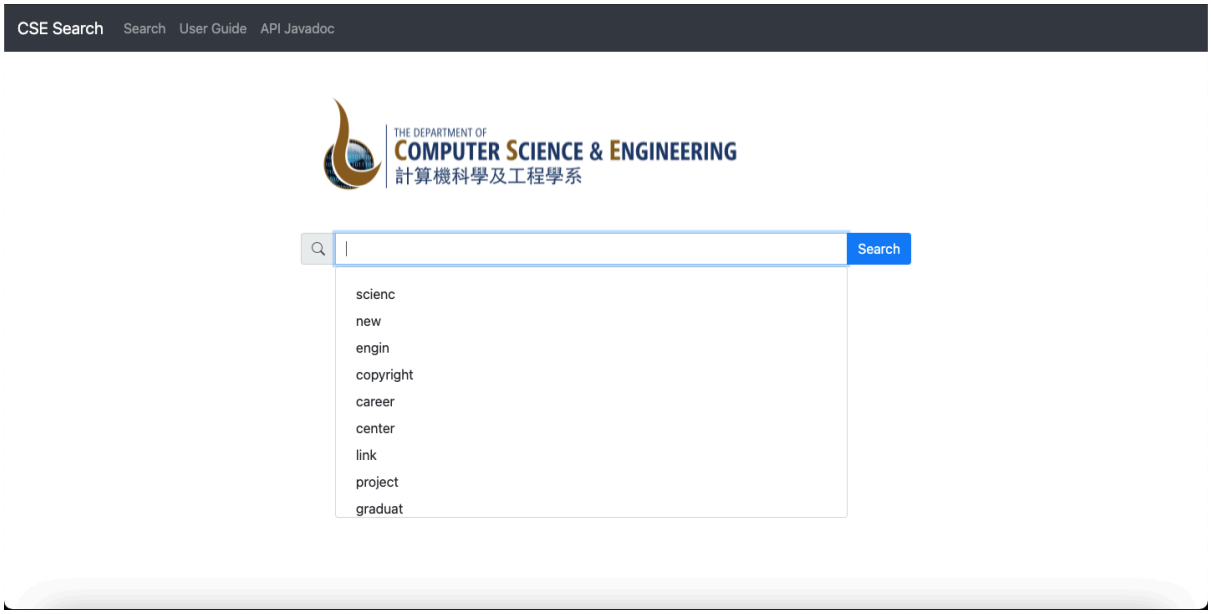


Figure 8: Query Input Page

A.3.2 Waiting for Result

When the program is continuously running to find relevant web pages, Figure 9 will be shown. In this stage, user simply has to patiently wait for the results retrieved by the search engine.

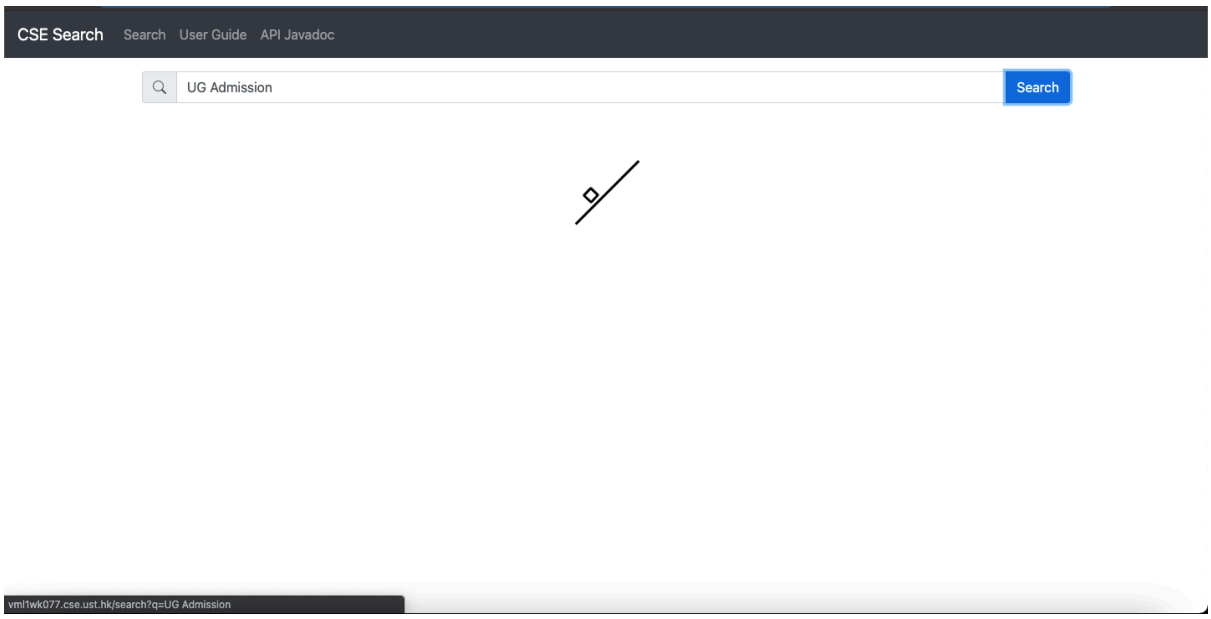


Figure 9: Loading Page

A.3.3 Viewing Result

If no relevant web pages are foundo, Figure 10 will be shown. In this case, user is suggested to modify their inputted query so as to obtain relevant results.

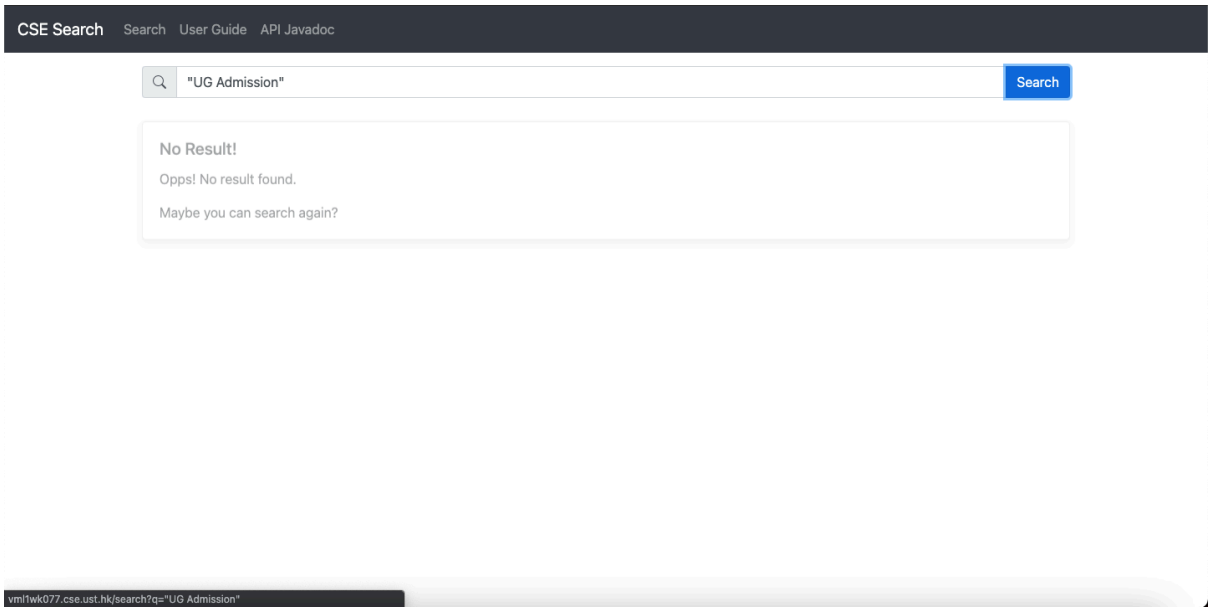


Figure 10: Result Page 1

If relevant web pages are found, the results will be displayed in a format as shown in Figure 11. For each relevant results found, user can view the similarity score as well as other basic information including page title, url, last modification date, size of page, keywords and their respective frequencies and child links. User can directly click on the url and child links if one is interested to know more about the web page. To view 5 more child links, user can click on `[Show more...]` and by recursively clicking on that, user will be able to view all related child links.

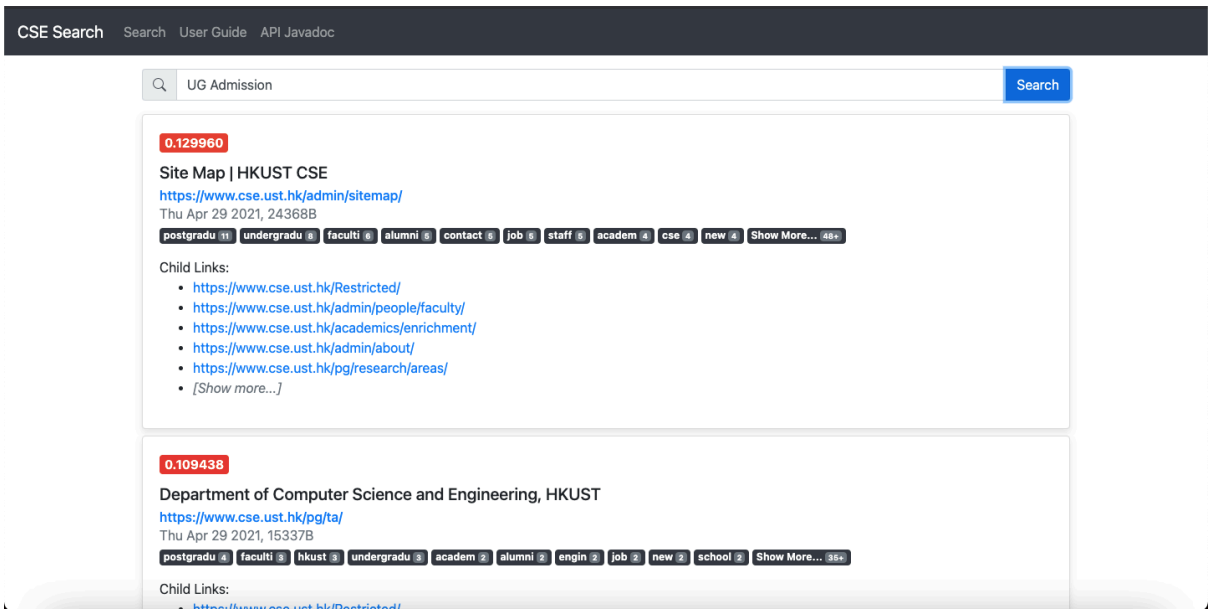


Figure 11: Result Page 2

A.4 Advanced Search

We had implemented phrase search and extended boolean model in our search engine. Here, will be a guide on how to specify these two searches.

A.4.1 Phrase Search

To specify phrase search, user has to use double quotes to quote the phrase. Figure 12 is an example illustrating the use of phrase search when inputting search query.

Figure 12: Phrase Search Example

A.4.2 Extended Boolean Model

As for specifying boolean queries, user has to use terms “AND” or “OR” according to one’s own preference. Figures 13 and 14 are examples illustrating the input of boolean search query.

Figure 13: Boolean Search AND Example

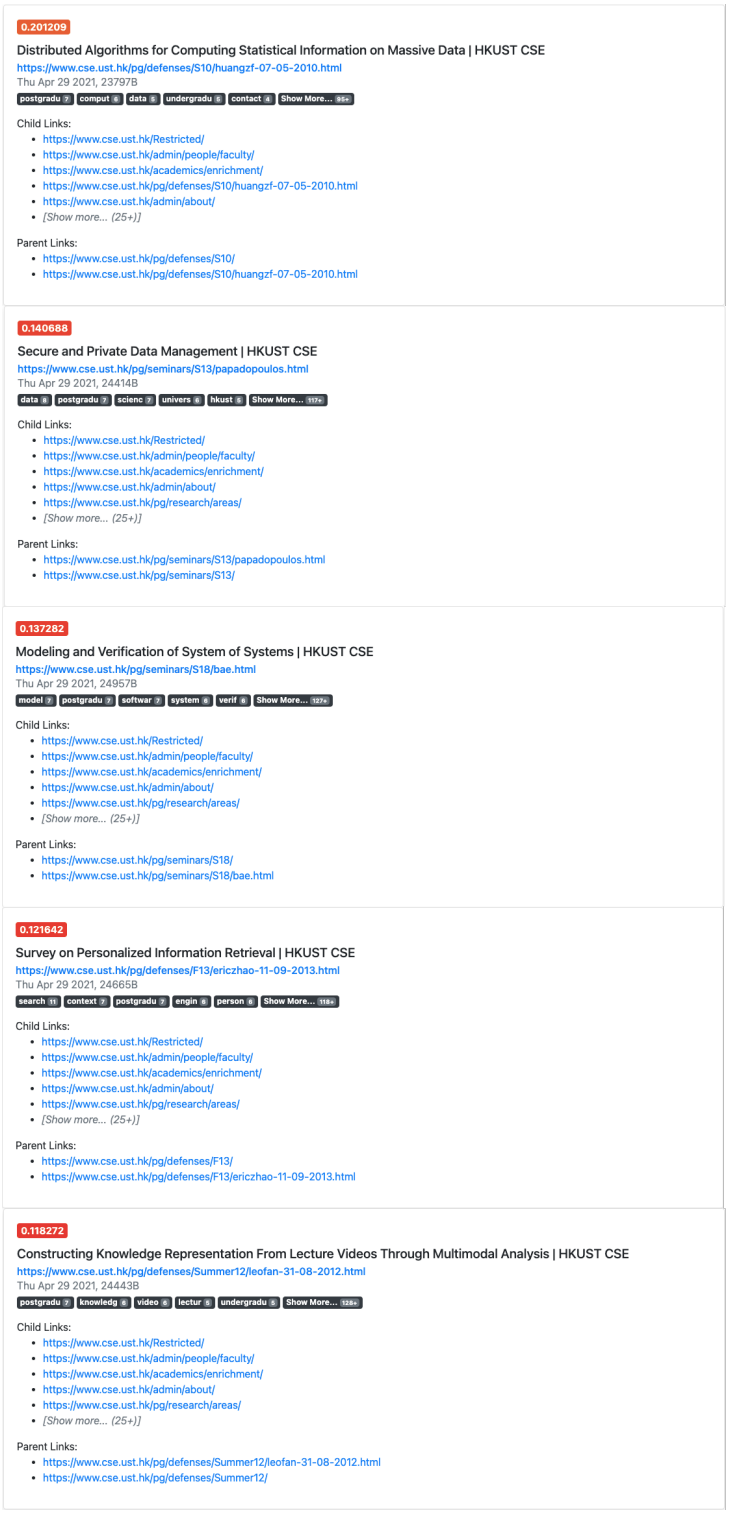
Figure 14: Boolean Search OR Example

B Appendix

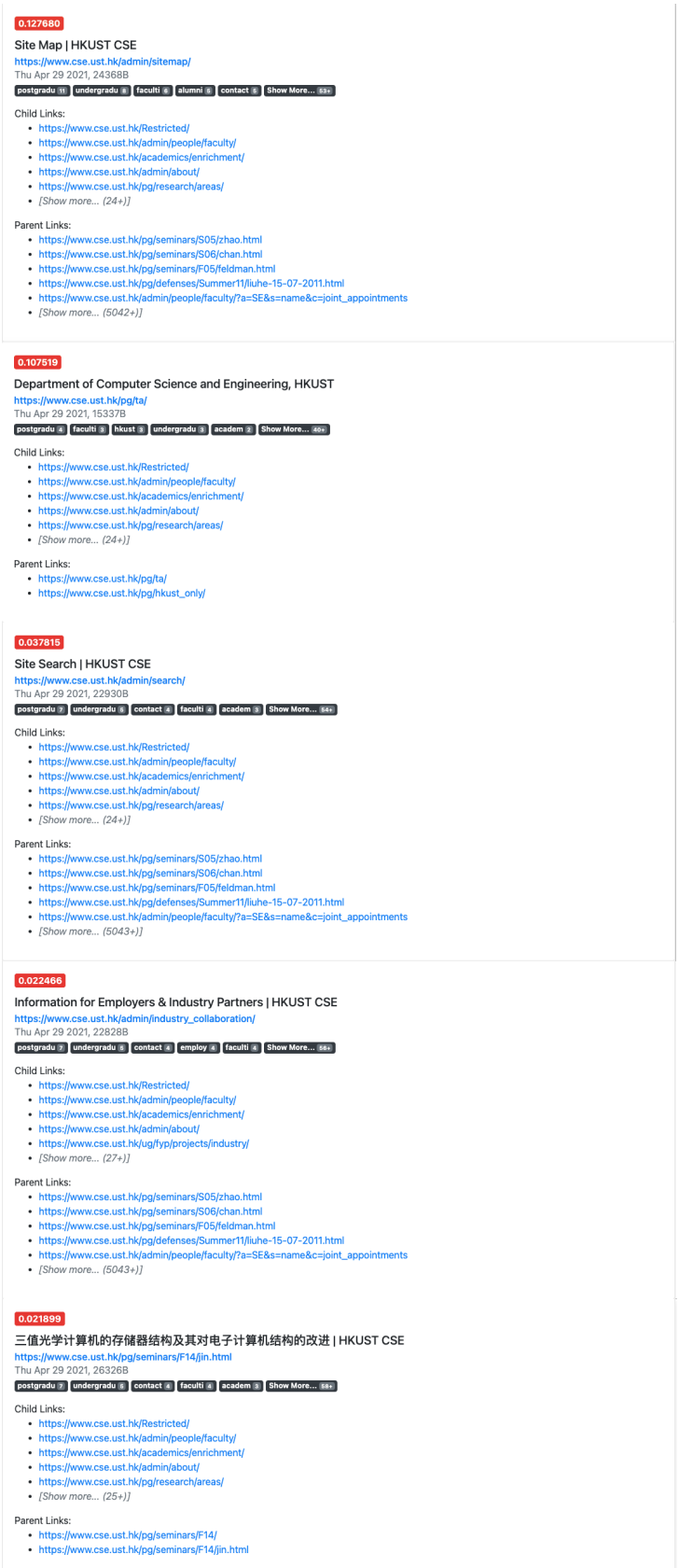
B.1 Search Result of Performance Evaluation Test

Below shows the screen captures of the search results in the three tests that we have conducted.

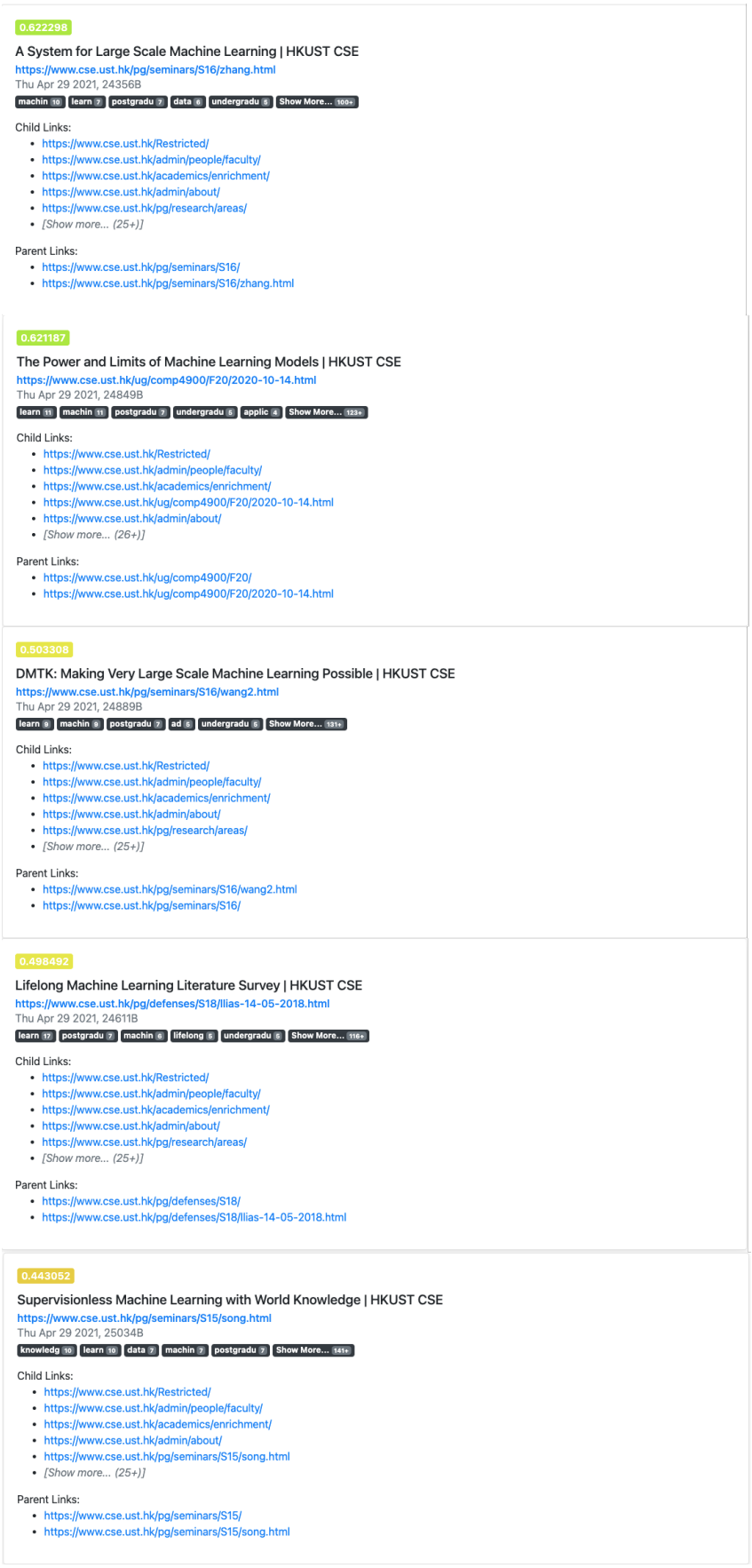
B.1.1 1st Test



B.1.2 2nd Test


0.127680
Site Map | HKUST CSE
https://www.cse.ust.hk/admin/sitemap/
Thu Apr 29 2021, 24368B
postgradu 11 undergradu 8 faculti 6 alumni 5 contact 5 Show More... 53+
Child Links:
https://www.cse.ust.hk/Restricted/
https://www.cse.ust.hk/admin/people/faculty/
https://www.cse.ust.hk/academics/enrichment/
https://www.cse.ust.hk/admin/about/
https://www.cse.ust.hk/pg/research/areas/
[Show more... (24+)]
Parent Links:
https://www.cse.ust.hk/pg/seminars/S05/zhao.html
https://www.cse.ust.hk/pg/seminars/S06/chan.html
https://www.cse.ust.hk/pg/seminars/F05/feldman.html
https://www.cse.ust.hk/pg/defenses/Summer11/liuhe-15-07-2011.html
https://www.cse.ust.hk/admin/people/faculty/?a=SE&s=name&c=joint_appointments
[Show more... (5042+)]
0.107519
Department of Computer Science and Engineering, HKUST
https://www.cse.ust.hk/pg/ta/
Thu Apr 29 2021, 15337B
postgradu 4 faculti 3 hkust 3 undergradu 3 academ 3 Show More... 40+
Child Links:
https://www.cse.ust.hk/Restricted/
https://www.cse.ust.hk/admin/people/faculty/
https://www.cse.ust.hk/academics/enrichment/
https://www.cse.ust.hk/admin/about/
https://www.cse.ust.hk/pg/research/areas/
[Show more... (24+)]
Parent Links:
https://www.cse.ust.hk/pg/ta/
https://www.cse.ust.hk/pg/hkust_only/
0.037815
Site Search | HKUST CSE
https://www.cse.ust.hk/admin/search/
Thu Apr 29 2021, 22930B
postgradu 7 undergradu 6 contact 4 faculti 4 academ 3 Show More... 54+
Child Links:
https://www.cse.ust.hk/Restricted/
https://www.cse.ust.hk/admin/people/faculty/
https://www.cse.ust.hk/academics/enrichment/
https://www.cse.ust.hk/admin/about/
https://www.cse.ust.hk/pg/research/areas/
[Show more... (24+)]
Parent Links:
https://www.cse.ust.hk/pg/seminars/S05/zhao.html
https://www.cse.ust.hk/pg/seminars/S06/chan.html
https://www.cse.ust.hk/pg/seminars/F05/feldman.html
https://www.cse.ust.hk/pg/defenses/Summer11/liuhe-15-07-2011.html
https://www.cse.ust.hk/admin/people/faculty/?a=SE&s=name&c=joint_appointments
[Show more... (5043+)]
0.022466
Information for Employers & Industry Partners | HKUST CSE
https://www.cse.ust.hk/admin/industry_collaboration/
Thu Apr 29 2021, 22828B
postgradu 7 undergradu 6 contact 4 employ 4 faculti 4 Show More... 58+
Child Links:
https://www.cse.ust.hk/Restricted/
https://www.cse.ust.hk/admin/people/faculty/
https://www.cse.ust.hk/academics/enrichment/
https://www.cse.ust.hk/admin/about/
https://www.cse.ust.hk/ug/fyp/projects/industry/
[Show more... (27+)]
Parent Links:
https://www.cse.ust.hk/pg/seminars/S05/zhao.html
https://www.cse.ust.hk/pg/seminars/S06/chan.html
https://www.cse.ust.hk/pg/seminars/F05/feldman.html
https://www.cse.ust.hk/pg/defenses/Summer11/liuhe-15-07-2011.html
https://www.cse.ust.hk/admin/people/faculty/?a=SE&s=name&c=joint_appointments
[Show more... (5043+)]
0.021899
三值光学计算机的存储器结构及其对电子计算机结构的改进 | HKUST CSE
https://www.cse.ust.hk/pg/seminars/F14/jin.html
Thu Apr 29 2021, 26326B
postgradu 7 undergradu 6 contact 4 faculti 4 academ 3 Show More... 58+
Child Links:
https://www.cse.ust.hk/Restricted/
https://www.cse.ust.hk/admin/people/faculty/
https://www.cse.ust.hk/academics/enrichment/
https://www.cse.ust.hk/admin/about/
https://www.cse.ust.hk/pg/research/areas/
[Show more... (25+)]
Parent Links:
https://www.cse.ust.hk/pg/seminars/F14/
https://www.cse.ust.hk/pg/seminars/F14/jin.html

B.1.3 3rd Test


0.622298
A System for Large Scale Machine Learning | HKUST CSE
https://www.cse.ust.hk/pg/seminars/S16/zhang.html
Thu Apr 29 2021, 24356B
machin 10 learn 7 postgradu 7 data 6 undergradu 5 Show More... 100+
Child Links:
https://www.cse.ust.hk/Restricted/
https://www.cse.ust.hk/admin/people/faculty/
https://www.cse.ust.hk/academics/enrichment/
https://www.cse.ust.hk/admin/about/
https://www.cse.ust.hk/pg/research/areas/
[Show more... (25+)]
Parent Links:
https://www.cse.ust.hk/pg/seminars/S16/
https://www.cse.ust.hk/pg/seminars/S16/zhang.html
0.621187
The Power and Limits of Machine Learning Models | HKUST CSE
https://www.cse.ust.hk/ug/comp4900/F20/2020-10-14.html
Thu Apr 29 2021, 24849B
learn 11 machin 11 postgradu 7 undergradu 5 applic 4 Show More... 123+
Child Links:
https://www.cse.ust.hk/Restricted/
https://www.cse.ust.hk/admin/people/faculty/
https://www.cse.ust.hk/academics/enrichment/
https://www.cse.ust.hk/ug/comp4900/F20/2020-10-14.html
https://www.cse.ust.hk/admin/about/
[Show more... (26+)]
Parent Links:
https://www.cse.ust.hk/ug/comp4900/F20/
https://www.cse.ust.hk/ug/comp4900/F20/2020-10-14.html
0.503308
DMTK: Making Very Large Scale Machine Learning Possible | HKUST CSE
https://www.cse.ust.hk/pg/seminars/S16/wang2.html
Thu Apr 29 2021, 24889B
learn 9 machin 9 postgradu 7 ad 5 undergradu 5 Show More... 131+
Child Links:
https://www.cse.ust.hk/Restricted/
https://www.cse.ust.hk/admin/people/faculty/
https://www.cse.ust.hk/academics/enrichment/
https://www.cse.ust.hk/admin/about/
https://www.cse.ust.hk/pg/research/areas/
[Show more... (25+)]
Parent Links:
https://www.cse.ust.hk/pg/seminars/S16/wang2.html
https://www.cse.ust.hk/pg/seminars/S16/
0.498492
Lifelong Machine Learning Literature Survey | HKUST CSE
https://www.cse.ust.hk/pg/defenses/S18/llias-14-05-2018.html
Thu Apr 29 2021, 24611B
learn 17 postgradu 7 machin 6 lifelong 5 undergradu 5 Show More... 116+
Child Links:
https://www.cse.ust.hk/Restricted/
https://www.cse.ust.hk/admin/people/faculty/
https://www.cse.ust.hk/academics/enrichment/
https://www.cse.ust.hk/admin/about/
https://www.cse.ust.hk/pg/research/areas/
[Show more... (25+)]
Parent Links:
https://www.cse.ust.hk/pg/defenses/S18/
https://www.cse.ust.hk/pg/defenses/S18/llias-14-05-2018.html
0.443052
Supervisionless Machine Learning with World Knowledge | HKUST CSE
https://www.cse.ust.hk/pg/seminars/S15/song.html
Thu Apr 29 2021, 25034B
knowledg 10 learn 10 data 7 machin 7 postgradu 7 Show More... 141+
Child Links:
https://www.cse.ust.hk/Restricted/
https://www.cse.ust.hk/admin/people/faculty/
https://www.cse.ust.hk/academics/enrichment/
https://www.cse.ust.hk/admin/about/
https://www.cse.ust.hk/pg/seminars/S15/song.html
[Show more... (25+)]
Parent Links:
https://www.cse.ust.hk/pg/seminars/S15/
https://www.cse.ust.hk/pg/seminars/S15/song.html