COMP4321 Search Engine

Project Report

Chan Tony Yuen Yeung, 20864385

Cheng Yan Hei, 20865872

WU, Man Yui, 20769755

Date: 03/05/2024

COMP4321 Search Engine

Overall Design of the System

Folder Structure

The project is organized into the following directories:

1. **data**: Contains the indexed database stored in the "database.db" file, utilizing the Java JDBM library. Additionally, it houses project documentation such as the Database Design document and the "stopwords.txt" file.
2. **docs**: Holds project documentation, including design documents and guides.
3. **lib**: Stores third-party libraries used in the project.
4. **src**: Divided into backend and frontend components, where backend code handles web crawling and indexing, while frontend code encompasses the search engine interface implemented using Apache Tomcat.

Backend: Web Crawler and Indexer (Java)

The backend of the system is implemented in Java, utilizing the following components:

1. **IRUtilities**: Package containing the Porter algorithm used for word stemming.
2. **StopStem**: Class responsible for stop word removal and word stemming.
3. **Spider**: Class used for crawling web pages and extracting basic information such as headers and body content.
4. **Indexer**: Class responsible for handling words, including calculating word frequency, and managing the indexed database.
5. **CrawlandIndex**: Central function that combines the web crawler and indexer to process page information.
6. **TestProgram**: Main testing program that controls the "CrawlandIndex" function and manages the database location, including copying it to the frontend directory. (You can set the maximum number of pages and the starting website here)

Frontend: Search Engine (Apache Tomcat)

The frontend of the system is implemented using Apache Tomcat, with the following components:

1. **apis**: Contains the "getData.jsp" file, responsible for handling input from the search engine, cleaning, stopping, and stemming the input. It retrieves relevant indexed words from the database, calculates final frequencies, and returns a list of pages with detailed information and sorted page IDs.
2. **html**: Holds frontend HTML elements used in the search engine page.
3. **images**: Stores the rectangular logo of the search engine.
4. **utils**: Contains JavaScript code responsible for handling button click events, tab navigation, and generating query results.

5. **WEB-INF**: Includes static materials used in the search engine result:
    a. **classes**: Holds the Porter algorithm for stopping and stemming the input.
    b. **databases**: Contains the "database.db" file for storing indexed results, which is the same as the one used in the backend.
    c. **lib**: Stores libraries used in the frontend, mirroring those used in the backend.
6. **index.jsp**: The main page of the frontend, containing code for the search engine interface.
7. **styles.css**: CSS styling code used to style the frontend interface.

File Structures Used in the Index Database

The index database utilizes the jdbm1.0 storage mechanism, storing data in a (Key, Value) format, with "|" and "," used as separators within values. A single database file is used, as all eight databases are included as objects within a single "Record Manager."

Database Design

We use true data types in the database design but implement keys and values as String types for simplicity. Here's the database design:

PageInfo

Description: Contains information about each page.

Field Name	Data Type	Description
PageID	Integer	Unique key ID for pages
PageTitle	String	Title of the page
URL	String	URL of the page
LastModificationDate	String	Date of the last modification
SizeofPage	Integer	Size of the page
MaxFreq	Int	Maximum Term Frequency
pageMag	Double	Page Magnitude

Data:

<table>
<tr><td>1</td><td>COMP4321 Project Website|www.testing.net|11/03/2024|96|20|32.5</td></tr>
</table>

PageURLMapping

Description: Maps URL to their respective PageID.

Field Name	Data Type	Description
URL	String	URL of the page
PageID	Integer	Unique key ID for pages

Data:

www.testing.net	1

PageParentMapping

Description: Maps child links to their respective parent pages.

Field Name	Data Type	Description
PageID	Integer	ID of the parent page
ParentID	Integer	ID of the parent pages

Data:

3	1, 2

PageChildMapping

Description: Maps child links to their respective parent pages.

Field Name	Data Type	Description
PageID	Integer	ID of the parent page
ChildID	Integer	ID of the child pages

Data:

1	2, 3

InvertedBodyWord

Description: Stores inverted index for words found in the body of pages.

Field Name	Data Type	Description
WordID	Integer	Unique identifier for body words
PageID	Integer	ID of the page containing the word
Frequency	Integer	Frequency of the word in the body
TFIDF/Max	Double	TF-IDF/Max value of the word

Data:

1	1|5|2.1, 2|4|1.8

BodyWordMapping

Description: Maps body words to the pages they appear in.

Field Name	Data Type	Description
PageID	Integer	ID of the page
WordID	Integer	ID of the body word
Frequncy	Integer	Frequency of the word
Word	String	The Word

Data:

1	1|3|apple, 3|4|egg, 6|5|test,

InvertedTitleWord

Description: Stores inverted index for words found in the title of pages.

Field Name	Data Type	Description
TitleWordID	Integer	Unique identifier for title words
PageID	Integer	ID of the page containing the word

Frequency	Integer	Frequency of the word in the title
TFIDF	Double	TF-IDF value of the word

Data:

<table>
<tr><td>3</td><td>1|5|2.1, 2|4|1.8</td></tr>
</table>

WordMapping

Description: Maps body word IDs to their respective words.

Field Name	Data Type	Description
Word	String	Body word
WordID	Integer	ID of the body word

Data:

Apple	1

Algorithms Used

Result Ranking

To rank search results, we employ a weighted sum approach. We calculate the cosine similarity of both the heading and the body of each page, assigning a weight of 25 to the heading and 10 to the body. This weighted sum helps prioritize pages that closely match the search query in both their heading and body content.

Breadth-First Search (BFS)

For web crawling and indexing, we utilize a Breadth-First Search algorithm. Starting from the parent node, we systematically discover new child nodes and add them to a queue if they haven't been visited before. We maintain a separate array to track visited nodes. Once all information from the current node is indexed, we visit the first node in the queue, repeating this process until there are no more nodes in the queue or the maximum number of visits is reached (in our project, set to 300 pages).

Text Processing (Stop Stem and N-gram)

In text processing, we implement stop word removal, word stemming, and extraction of n-grams. Using the HTML parser introduced in our lab, we extract words from the body of each page. The “extractWords” function in Spider.java facilitates this process. We utilize a combination of “String Bean” and “String Tokenizer” to create a moving window of three words (i.e. this, next and next_2). By applying “nextToken” and “hasMoreTokens”, we generate combinations of stemmed words to be indexed, including 2-grams and 3-grams. Additionally, we leverage “prev” and “prev_2” to capture all possible positions of the words within the text.

Installation Procedure

Prerequisites

OpenJDK version: 21.0.2

Building the Project

To compile each Java file, follow the commands below (Please make sure you are in the src directory):

1. **StopStem**:

```
javac "StopStem.java"
```

2. **Indexer**:

```
javac -cp "../lib/jdbm-1.0.jar;." "Indexer.java"
```

3. **Spider**:

```
javac -cp "../lib/htmlparser.jar;." "Spider.java"
```

4. **CrawlandIndex**:

```
javac -cp "../lib/htmlparser.jar;../lib/jdbm-1.0.jar;." "CrawlandIndex.java"
```

5. **TestProgram**:

```
javac "TestProgram.java"
```

Executing the Program:

After compiling the Java files, execute the TestProgram to initiate the web crawling and indexing process using the following command:

```
java -cp "../lib/htmlparser.jar;../lib/jdbm-1.0.jar;." "TestProgram"
```

After executing the TestProgram, the following files will be generated:

"database.db" in the backend folder, "../data/" and in the frontend folder, "./apache-tomcat-10.1.20/webapps/comp4321/WEB-INF/database/".

(The database.db should be copied to the frontend folder automatically by the TestProgram. If not, please copy it manually for the frontend to work properly.)

Running the Frontend:

Before running the frontend, set up the system environment variables for Apache Tomcat as follows:

KEY	VALUE
CATALINA_HOME	{path to this project}\COMP4321-Crawler\src\apache-tomcat-10.1.20

JAVA_HOME	{path to your JDK} (e.g. "C:\Program Files\Eclipse Adoptium\jdk-21.0.2.13-hotspot")

After setting up the system environment variables, start the Apache Tomcat server by running the following command (Please make sure you are in the src directory):

```
apache-tomcat-10.1.20/bin/startup.bat
```

Then, open a web browser and navigate to the following URL to access the frontend: http://localhost:8080/comp4321/

Highlights of Features Beyond the Required Specification

Additional Functionalities

1. Operation on Search Query Result

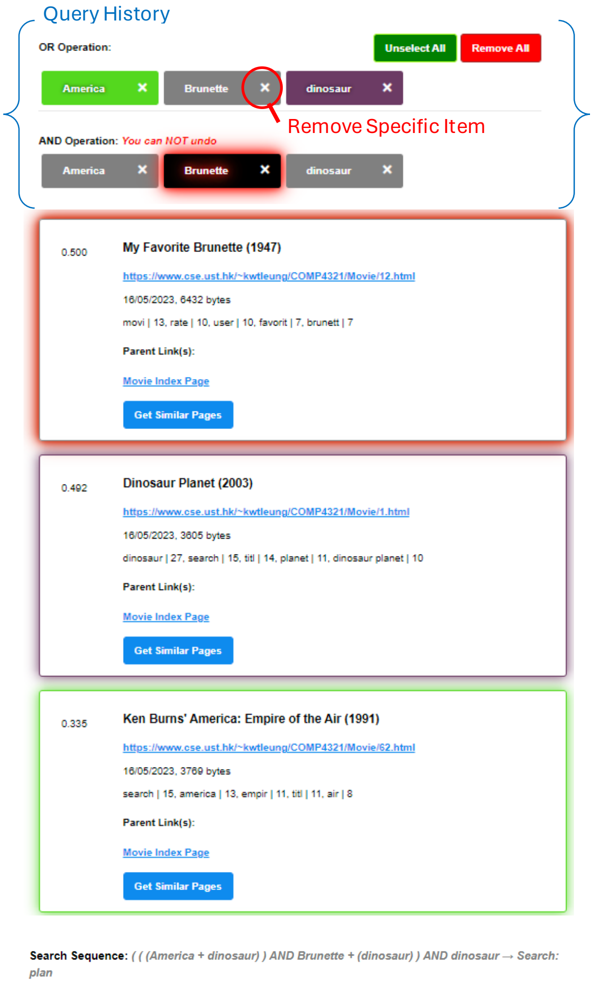
The user's search results can be conveniently stored in local storage, which remain in the browser after the session ends, allowing them to preview and perform multiple AND/OR operations to filter the results.

The resulting content is visually labelled with different colors to indicate which operation they belong to.

Additionally, users have the ability to search within the filtered content.

A search sequence is also shown to aid users in keeping track of those operations applied.

Note: Click "Remove All" to remove those query results from your local storage.



0.500 My Favorite Brunette (1947)

https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/12.html

16/05/2023, 6432 bytes

movi | 13, rate | 10, user | 10, favorit | 7, brunett | 7

Parent Link(s):

Movie Index Page

Get Similar Pages

0.492 Dinosaur Planet (2003)

https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/1.html

16/05/2023, 3605 bytes

dinosaur | 27, search | 15, titl | 14, planet | 11, dinosaur planet | 10

Parent Link(s):

Movie Index Page

Get Similar Pages

0.335 Ken Burns' America: Empire of the Air (1991)

https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/62.html

16/05/2023, 3769 bytes

search | 15, america | 13, empir | 11, titl | 11, air | 8

Parent Link(s):

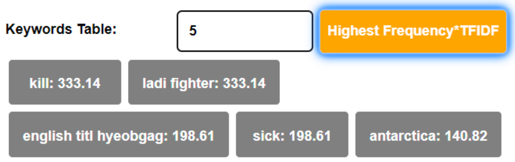
Movie Index Page

Get Similar Pages

Search Sequence: *(((America + dinosaur)) AND Brunette + (dinosaur)) AND dinosaur → Search: plan*

2. Keywords Table

The keywords table provides a list of stemmed keywords with the highest frequency or TFIDF scores and users then submit these keywords directly into the query.



Keywords Table: 5

kill: 333.14
ladi fighter: 333.14
english titl hyeobgag: 198.61
sick: 198.61
antarctica: 140.82

Besides, users can also modify the number of keywords displayed according to their specific needs.

3. Get Similar Pages Button

The "Get Similar Pages" button is attached to each of the resulted page, it automatically submits the most frequent word found within that specific page as a new query.

User Experience Enhancements

Building upon the strengths discussed in the Conclusion section, our system offers several user experience enhancements beyond the required specifications:

1. Smooth Searching Experience

Efficient Query Processing: Users benefit from rapid query processing, ensuring quick retrieval of relevant search results and a seamless searching experience.

2. Short Scrolling Area

Tabbed Navigation: Organizing search results into tabs with a limited number of results per tab minimizes scrolling, improving user efficiency and satisfaction.

Optimized Display: Each tab presents a concise list of results, enabling users to quickly locate desired information without excessive scrolling.

3. Simplistic Style

Clean and Organized Design: The simplistic and organized design promotes focus and clarity, with streamlined elements and minimal distractions enhancing user comprehension.

4. Contrast on Important Features

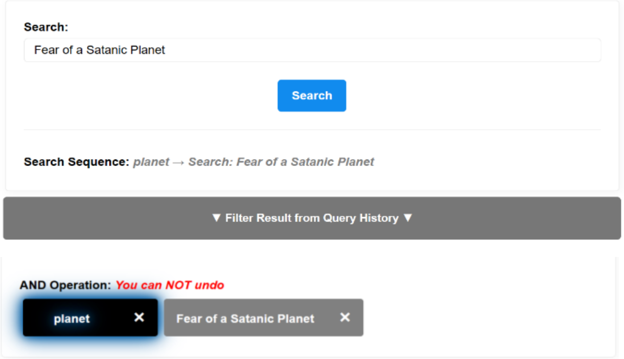
Effective Highlighting: Key features and relevant information are highlighted effectively, drawing users' attention to critical elements and improving usability.

5. Enhanced Query Handling

Multi-Phrase Support: Users can input multiple phrases and keywords, with advanced processing mechanisms such as stop-word removal and word stemming, resulting in more accurate and relevant search results.

Testing of the Functions Implemented

1. Searching on top of previous result using AND operator



Search:

Fear of a Satanic Planet

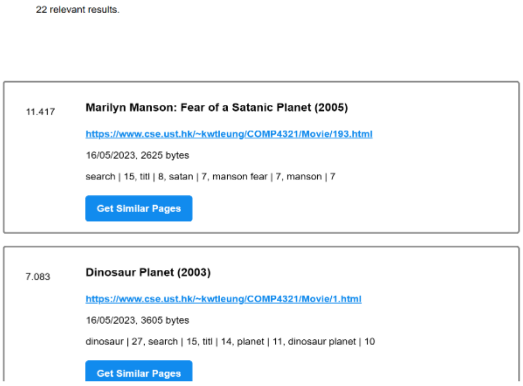
Search

Search Sequence: *planet → Search: Fear of a Satanic Planet*

▼ Filter Result from Query History ▼

AND Operation: *You can NOT undo*

planet ✕ Fear of a Satanic Planet ✕

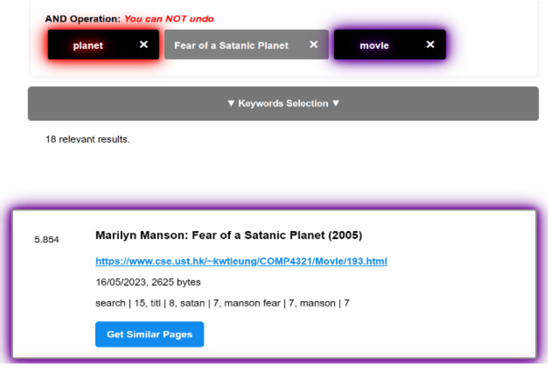


22 relevant results.

11.417 **Marilyn Manson: Fear of a Satanic Planet (2005)**
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/193.html
16/05/2023, 2625 bytes
search | 15, titl | 8, satan | 7, manson fear | 7, manson | 7
Get Similar Pages

7.083 **Dinosaur Planet (2003)**
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/1.html
16/05/2023, 3605 bytes
dinosaur | 27, search | 15, titl | 14, planet | 11, dinosaur planet | 10
Get Similar Pages

2. Combining the previous two results using AND operator



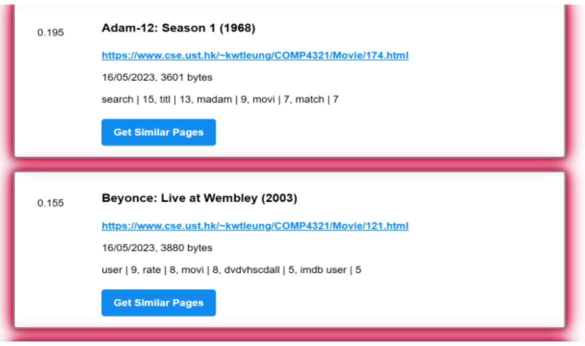
AND Operation: *You can NOT undo*

planet ✕ Fear of a Satanic Planet ✕ movie ✕

▼ Keywords Selection ▼

18 relevant results.

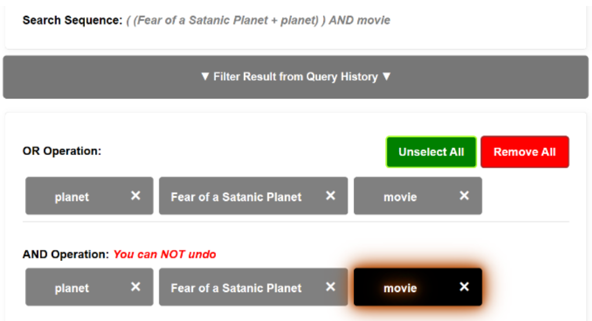
5.854 **Marilyn Manson: Fear of a Satanic Planet (2005)**
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/193.html
16/05/2023, 2625 bytes
search | 15, titl | 8, satan | 7, manson fear | 7, manson | 7
Get Similar Pages



0.195 **Adam-12: Season 1 (1968)**
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/174.html
16/05/2023, 3601 bytes
search | 15, titl | 13, madam | 9, movi | 7, match | 7
Get Similar Pages

0.155 **Beyonce: Live at Wembley (2003)**
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/121.html
16/05/2023, 3880 bytes
user | 9, rate | 8, movi | 8, dvdvhscdall | 5, imdb user | 5
Get Similar Pages

3. Combining the previous results using OR + AND operator



Search Sequence: *((Fear of a Satanic Planet + planet)) AND movie*

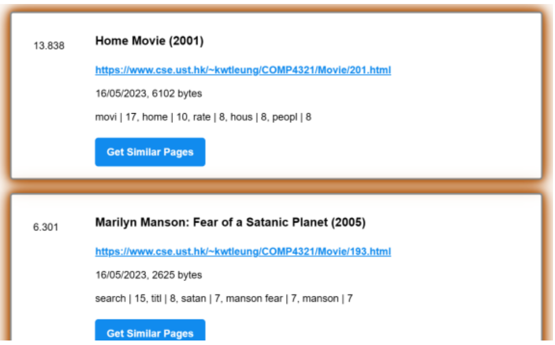
▼ Filter Result from Query History ▼

OR Operation: Unselect All Remove All

planet ✕ Fear of a Satanic Planet ✕ movie ✕

AND Operation: *You can NOT undo*

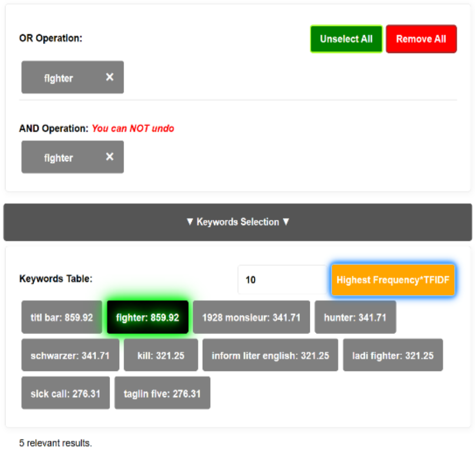
planet ✕ Fear of a Satanic Planet ✕ movie ✕



13.838 **Home Movie (2001)**
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/201.html
16/05/2023, 6102 bytes
movi | 17, home | 10, rate | 8, hous | 8, peopl | 8
Get Similar Pages

6.301 **Marilyn Manson: Fear of a Satanic Planet (2005)**
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/193.html
16/05/2023, 2625 bytes
search | 15, titl | 8, satan | 7, manson fear | 7, manson | 7
Get Similar Pages

4. Keywords Table (Highest TFIDF/Highest Frequency*TFIDF/Highest TFIDF)



OR Operation: Unselect All Remove All

fighter ✕

AND Operation: *You can NOT undo*

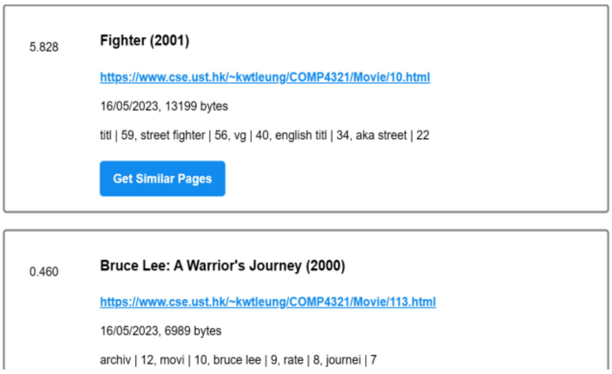
fighter ✕

▼ Keywords Selection ▼

Keywords Table: 10 Highest Frequency*TFIDF

titl bar: 859.92 fighter: 859.92 1928 monsieur: 341.71 hunter: 341.71
schwarzer: 341.71 kill: 321.25 inform liter english: 321.25 ladi fighter: 321.25
sick call: 276.31 taglin five: 276.31

5 relevant results.



5.828 **Fighter (2001)**
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/10.html
16/05/2023, 13199 bytes
titl | 59, street fighter | 56, vg | 40, english titl | 34, aka street | 22
Get Similar Pages

0.460 **Bruce Lee: A Warrior's Journey (2000)**
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/113.html
16/05/2023, 6989 bytes
archiv | 12, movi | 10, bruce lee | 9, rate | 8, journei | 7

Conclusion

Strengths

User Experience (UX)

Smooth Searching Experience: The system offers rapid query processing, ensuring users receive search results promptly.

Short Scrolling Area: Organizing results into tabs, with five results per tab, minimizes scrolling and aids in locating relevant pages efficiently.

Easy Tab Navigation: Users can effortlessly navigate between tabs to access previously important results.

User Interaction (UI)

Simplistic Style: The UI features clean and organized elements, enhancing usability and visual appeal.

Contrast on Important Features: Key features are highlighted effectively, guiding users' attention to essential information.

Functionalities

Allow Input for Multiple Phrases and Keywords: Users can input multiple phrases and keywords, with the backend employing stop and stem mechanisms for efficient processing.

Weaknesses

Crawl and Index

Resource Intensive: The crawling and indexing process requires significant resources and time, impacting overall system efficiency.

Potential Loss of Child Links: Newly crawled pages may not revisit indexed pages, leading to missing child links in the records (Child links are temporally in the URL format and will be changed to Page ID that are indexed. If the child pages are not indexed in the iteration, they are gone in the parent-child relationship).

Result Ranking

Omission of Important Features: Some crucial features, such as metadata and original words, are neglected in the result ranking process.

Stop and Stem

Inclusion of Excessive Words and N-gram Phrases: The stop and stem mechanism may include unnecessary words and n-gram phrases, leading to cluttered results.

Re-Implementation

Whole System Setup

Programming Language: Transitioning to Python with Django backend and SQLite database, and potentially React for frontend, offers familiarity and access to a broader range of libraries.

Reasoning: Python's extensive library ecosystem and streamlined debugging process could enhance development efficiency and performance.

Indexing Functions

Optimization Requirement: The indexing functions require optimization to improve runtime efficiency and resource utilization.

Interesting Features

Implicit & Explicit Features

User Feedback Collection: Implementing implicit and explicit feedback mechanisms to build user profiles for personalized search experiences.

Page Rank based on Category

Categorization Algorithm: Introducing a page ranking algorithm based on category tags, akin to Google's approach, to enhance search quality and personalization.

Input Analysis using LLM Model

Advanced Input Analysis: Leveraging Language Model (LLM) to analyze user input strings and adjust weighting dynamically for improved search relevance.

Contribution

Tony (33.3%)	Backend Crawling, Indexing, Database Design
Cathy (33.3%)	Part of the Frontend, Report, Algorithms & Formula
York (33.3%)	Part of the Frontend, Video, Additional Features