COMP4321 Group 27 Project Final Report

Cheung Tuen King (20983959)

Li Ching Ho (21017111)

Chan Yuk Ki (20852215)

Demo Video URL: https://youtu.be/X005Ww9pGyY

1. Overall design of the system

The web-based search engine system is designed to provide efficient crawling, indexing, and searching of webpages. The design consists of four major components:

- **Crawler (Spider)**:
  Developed in Python 3.11.5, the crawler is responsible for fetching web pages from specified URLs using a breadth-first search (BFS) strategy. This approach ensures that all accessible pages from a starting point are systematically retrieved and indexed. The crawler extracts essential data, including page titles, URLs, and hyperlinks, while maintaining a structured format for easy processing. It handles cyclic links by preventing ancestors recorded as child links to avoid redundancy.

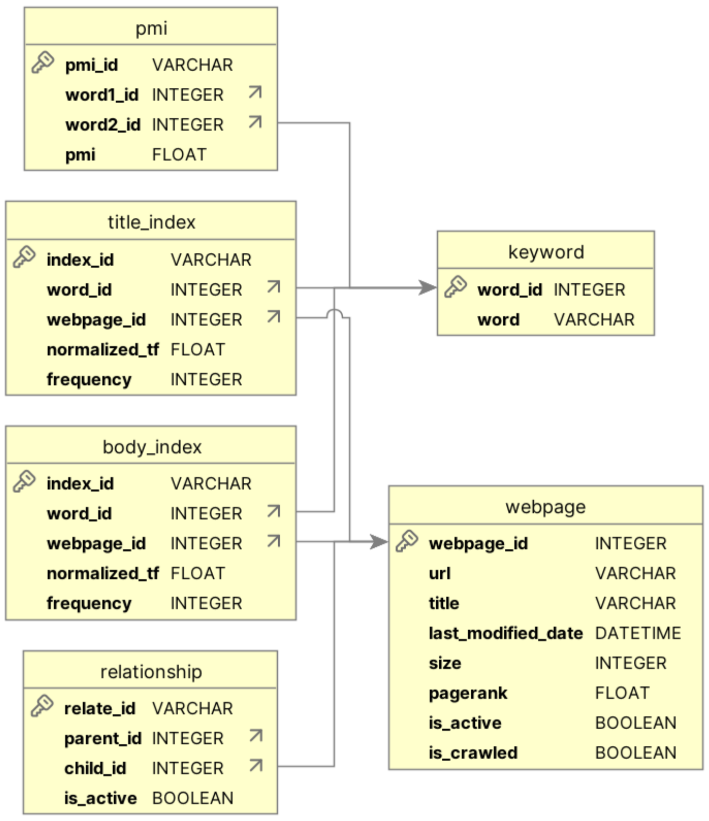
- **Indexer**:
  Once the crawler fetches the web pages, the indexer processes this data to extract meaningful keywords. Utilizing the Natural Language Toolkit (NLTK), the indexer removes stop words and applies stemming using the PorterStemmer algorithm. This process reduces words to their root forms, enhancing the efficiency of keyword searches. The indexer maps keywords to their corresponding webpage IDs, allowing for rapid lookups during search queries. The indexed data is stored in an SQLite database, ensuring persistence and easy retrieval.

- **Search Engine**:
  The search engine component compares user queries against the indexed data. It employs a cosine similarity measure to rank results based on relevance. The system integrates advanced features such as PageRank, which further refines search results by considering the importance of webpages. The search engine's architecture is designed to handle various query types, including simple keyword searches and complex phrase queries, ensuring a comprehensive search capability.

- **Web Interface**:
  The user interface, built with Node.js and Next.js, provides an intuitive platform for user interactions. It allows users to input queries and view results in a clear format. The interface includes features such as query suggestions, search history tracking, and advanced search options, enabling users to refine their searches. The design prioritizes user experience, ensuring that results are displayed in an organized manner, complete with metadata like page titles, URLs, and parent/child links.

2. File Structures Used in the Index Database

The system uses an SQLite database (project.db) with the following tables to manage indexing and relationships:



- **Webpage**: Stores URL-to-ID mappings, metadata (title, last modified date, size), and flags (is_active, is_crawled). The pagerank field precomputes PageRank scores for ranking.
- **Keyword**: Maps stemmed keywords to unique word_ids.
- **Relationship**: Tracks parent-child links between webpages to support BFS and cyclic link handling.
- **TitleIndex and BodyIndex**: Act as inverted files, storing term frequencies for title and body keywords.
- **PMI**: Stores precomputed Pointwise Mutual Information scores for keyword pairs to enhance query reformulation.

Rationale:

- **SQLite** was chosen for its lightweight nature and ease of integration with Python, replacing the suggested JDBM for better compatibility with the Python ecosystem.
- **Forward and inverted indexes** are implemented via TitleIndex and BodyIndex tables, enabling efficient TF-IDF calculations.
- **PMI table** supports bonus features like query expansion with co-occurring terms.

3. Algorithm Used

The web-based search engine employs several key algorithms to ensure efficient crawling, indexing, and retrieval of web pages. Each algorithm plays a crucial role in optimizing the search experience, enhancing both performance and relevance of the results.

- **Breadth-First Search (BFS)**:
  The crawler utilizes the BFS algorithm to fetch web pages systematically. Starting from a specified URL, BFS explores all reachable links layer by layer, ensuring that each page is visited in an organized manner. This method allows the crawler to maintain a queue of URLs to visit, processing each one before moving on to subsequent links. BFS is particularly effective for web crawling as it ensures that pages are indexed in the order they are discovered, which can help in capturing the most relevant and timely content. To prevent cyclic links, **ancestors of the webpages would not be recorded as child if they have a link to them.**

- **Keyword Extraction**:
  The indexer employs the Natural Language Toolkit (NLTK) for keyword extraction, which is essential for creating an effective search index. Initially, the indexer removes stop words. After that, the indexer applies the PorterStemmer function from NLTK, which implements Porter's algorithm to stem the remaining words. This process reduces words to their root forms, consolidating variations into a single representation. This stemming improves search accuracy and efficiency by ensuring that different forms of a word are treated as equivalent.
  For phrasal keyword extraction, the indexer utilizes the RAKE algorithm (Rapid Automatic Keyword Extraction). This method identifies multi-word phrases (2-3 words) that frequently occur together, enhancing the contextual relevance of search queries. By combining both single-word stemming and phrasal keyword extraction, the system effectively captures the essential terms that reflect the content of the web pages, thereby optimizing the indexing process for improved search results.

- **TF-IDF and Cosine Similarity:**
  Term weights in documents and queries are computed using TF-IDF. For documents:
  - **TitleTFIDF** and **BodyTFIDF** are calculated separately.
  - The final document weight combines these with a 7:3 ratio to prioritize title matches:

  $$TFIDF_{document}(t) = 0.7 \times TFIDF_{title}(t) + 0.3 \times TFIDF_{body}(t)$$

  Query vectors use the same weighting scheme, which can be expressed as:

  $$TFIDF_{query}(t) = TF_{query}(t) \times \left[0.7 \times IDF_{title}(t) + 0.3 \times IDF_{body}(t)\right]$$

  Cosine similarity measures the angle between query and document vectors to compute relevance scores. Scores are normalized to [0, 1] for display.

- **PageRank Integration:**
  PageRank scores are precomputed during indexing using an iterative algorithm (20 iterations). Each webpage's score is updated based on its incoming links and the rank of parent pages. Scores are min-max normalized and blended with cosine similarity to produce the modified score:

$$ModifiedScore = CosineSimilarity \times 0.7 + NormalizedPageRank \times 0.3$$

  This balances content relevance with link authority.

- **Relevance Feedback Mechanism:**
  User interactions (likes, clicks, dislikes) are tracked via cookies. Historical data is used to reformulate queries by computing weighted centroids of document vectors. For a query vector q_c, given $Liked$, $Disliked$, $Clicked$ are set of liked, disliked and clicked webpages in form of $(q,\ d)$ where d is the document vector of webpages and q is the query vector giving d as result, the reformulated vector q_c' is derived as:

$$Centroid(A) = \frac{\Sigma[CosSimilarity(q,\ q_c) \times d\,]}{\Sigma CosSimilarity(q,\ q_c)}\ \forall\ (q,\ d)\ \in\ A$$

$$q_c' = q_c + 0.5 \times Centroid(Liked\ \cup\ Clicked) - 0.25 \times Centroid(Disliked)$$

  where centroids are weighted by cosine similarity of each q to q_c. This dynamically adapts results to user preferences.

- **PMI-driven Query Expansion:**
  Pointwise Mutual Information (PMI) is precomputed for all keyword pairs during indexing, with the window size of entire title and body. For a query term a, co-occurring terms b with top PMI scores are added to the query. Co-occurring terms' TF-IDF is adjusted using:

$$PMI(a,\ b) = \frac{\min[TF(a),\ TF(b)] \times N}{TF(a) \times TF(b)}\ where\ N = sum\ of\ TF$$

$$WeightedPMI(a,\ b) = PMI_{title}(a,\ b) \times 0.7 + PMI_{body}(a,\ b) \times 0.3$$

$$TF_{query}(b) = 0.3 \times WeightedPMI(a,b) \times TF_{query}(a)$$

$$TFIDF_{query}(b) = TF_{query}(b) \times \left[0.7 \times IDF_{title}(b) + 0.3 \times IDF_{body}(b)\right]$$

- **Query Suggestion Algorithms:**
  Real-time suggestions combine:
  1. **Substring Matching**: The system retrieves keywords containing any query term as a substring. These terms are ranked by their occurrence probability in the database, with the top 5 displayed.
  2. **Historical Queries**: Ranked by cosine similarity to the current input.
  3. **Co-occurring Terms**: Suggested based on precomputed PMI.
  4. **Relevant Terms**: Derived from relevance feedback using liked, clicked and disliked pages.

- **Phrasal Search Support:**
  RAKE-extracted phrases and PMI scores enable phrase matching. While inputting query, phrases or keywords can be double quoted to ensure inclusion in all search results. During retrieval, phrases are treated as single units in query vectors, with TF-IDF computed for the entire phrase. This ensures accurate results for queries like "search engine" by prioritizing pages where the exact phrase appears.

By integrating these algorithms, the system achieves a balance of efficiency, relevance, and user-centric features, fulfilling both core requirements and advanced functionalities.

4. Installation procedure

4.1 Server Setup

1. Open a terminal.
2. At the project directory, run ***pip install -r requirements.txt*** in command line to install necessary libraries and packages.
3. Go to server folder by running ***cd server*** in command line
4. Run ***python spider.py*** to crawl 300 pages. This should take around 1-2 minutes.
5. Run ***python bonus.py*** to compute all bonus features. This should take less than 6 minutes.
6. Run ***uvicorn main:app --reload*** to start the server.
7. Record the server URL, which will be used in client setup.

4.2 Client Setup

1. Open another terminal (keeps server terminal running).
2. Install Typescript, Node.js and Next.js.
3. Go to client folder by running ***cd client*** in command line.
4. Open ***utils/api.ts*** file and replace ***api_url*** constant with the server URL.
5. Run ***npm run dev*** as **administrator** and click on the URL in the command line to view the webpage (in developer mode).

5. Highlight of features beyond the required specification

The system implements several advanced features to exceed baseline functionality.

- **Get Similar Pages** extracts the top 5 most frequent keywords (excluding stop words) from the selected page, reformulates the query, and triggers a new search. The viewed page is recorded with a relevance score of 1, influencing future relevance feedback.

- **PageRank integration** introduces link analysis into ranking, simulating how search engines prioritize authoritative pages. This required iterative computation of PageRank scores during indexing and storing them in the database, enabling real-time blending with TF-IDF scores during retrieval.

- **Relevance feedback mechanism** allows the system to adapt to user behavior. By tracking clicked, liked, and disliked pages, the engine reformulates queries using a weighted centroid of historical data. This feature, inspired by relevance feedback in information retrieval research, personalizes results without requiring explicit query modifications from the user.

- **Advanced search interface** supports Boolean operators (AND/OR/NOT) for title/body fields, date-range filters, and query joins. For example, users can search for pages where "machine learning" MUST appear in the title AND "neural networks" MAY appear in the body, modified between specific dates.

- **Subquery and merged search** allow combining results from multiple searches, with scores averaged across joined queries. Merged search allows up to 5 query input while subquery only allows 2 (original query and subquery) input.

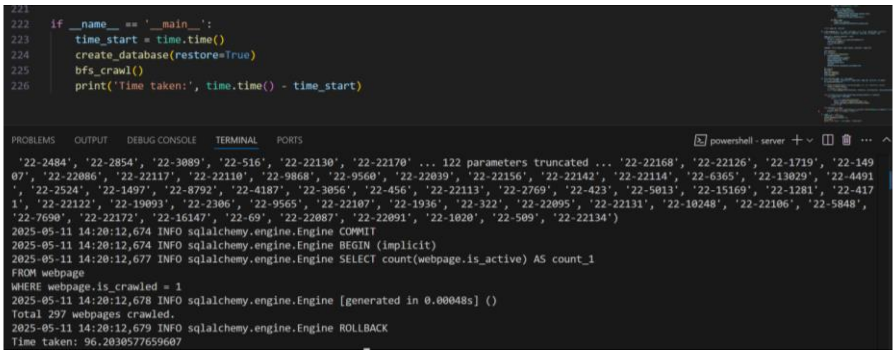
- **Query history** allows users access the **/history** page to review past searches, including query strings, dates, and displayed results (limited to viewed, liked or disliked pages).

- **Phrasal keyword extraction** using RAKE and **PMI-driven query expansion** significantly improve recall for multi-term queries. By precomputing PMI during indexing, the system efficiently identifies and suggests contextually related terms (e.g., expanding "AI" to "artificial intelligence algorithms"), bridging vocabulary gaps between user queries and indexed content. These features collectively create a robust, user-centric search experience.
- **Query suggestion** while inputting query and **keyword suggestion** while searching and selecting keywords allow personalized recommendation for user searching based on past queries and co-occurring terms.

6. Testing of the functions Implemented

Testing on Basic Crawling (spider.py)

Basic crawling should have 300 webpages crawled under 2 minutes (~96 seconds in this case).

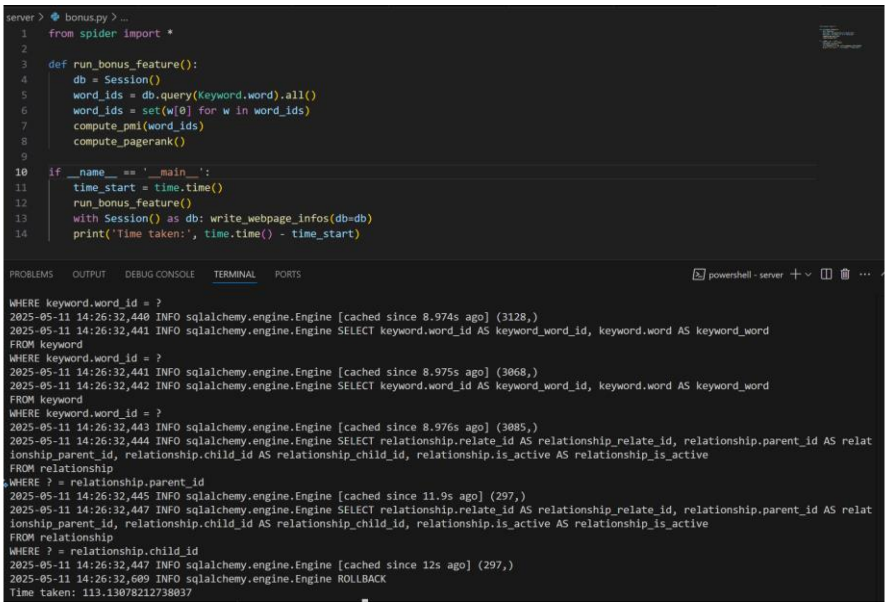


```python
if __name__ == '__main__':
    time_start = time.time()
    create_database(restore=True)
    bfs_crawl()
    print('Time taken:', time.time() - time_start)
```

```
 '22-2484', '22-2854', '22-3089', '22-516', '22-22130', '22-22170' ... 122 parameters truncated ... '22-22168', '22-22126', '22-1719', '22-149
07', '22-22086', '22-22117', '22-22110', '22-9868', '22-9560', '22-22039', '22-22156', '22-22142', '22-22114', '22-6365', '22-13029', '22-4491
', '22-2524', '22-1497', '22-8792', '22-4187', '22-3056', '22-456', '22-22113', '22-2769', '22-423', '22-5013', '22-15169', '22-1281', '22-417
1', '22-22122', '22-19093', '22-2306', '22-9565', '22-22107', '22-1936', '22-322', '22-22095', '22-22131', '22-10248', '22-22106', '22-5848',
'22-7690', '22-22172', '22-16147', '22-69', '22-22087', '22-22091', '22-1020', '22-509', '22-22134')
2025-05-11 14:20:12,674 INFO sqlalchemy.engine.Engine COMMIT
2025-05-11 14:20:12,674 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2025-05-11 14:20:12,677 INFO sqlalchemy.engine.Engine SELECT count(webpage.is_active) AS count_1
FROM webpage
WHERE webpage.is_crawled = 1
2025-05-11 14:20:12,678 INFO sqlalchemy.engine.Engine [generated in 0.00048s] ()
Total 297 webpages crawled.
2025-05-11 14:20:12,679 INFO sqlalchemy.engine.Engine ROLLBACK
Time taken: 96.2030577659607
```

Testing on Additional Feature Pre-computing (bonus.py)

After crawling, bonus feature including PageRank and PMI should be computed and stored in database. This may take less than 6 minutes (~113 seconds in this case). Spider_result.txt is shown to indicate the success of crawling.

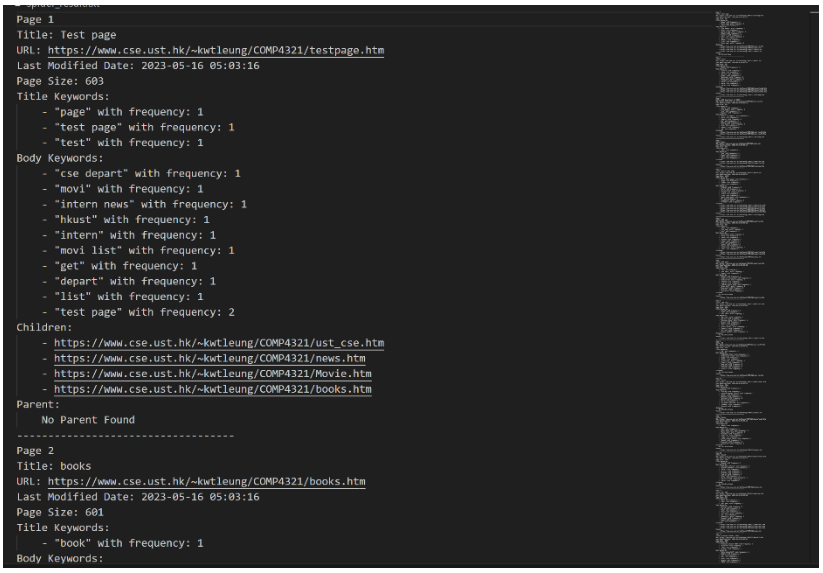


```python
from spider import *

def run_bonus_feature():
    db = Session()
    word_ids = db.query(Keyword.word).all()
    word_ids = set(w[0] for w in word_ids)
    compute_pmi(word_ids)
    compute_pagerank()

if __name__ == '__main__':
    time_start = time.time()
    run_bonus_feature()
    with Session() as db: write_webpage_infos(db=db)
    print('Time taken:', time.time() - time_start)
```

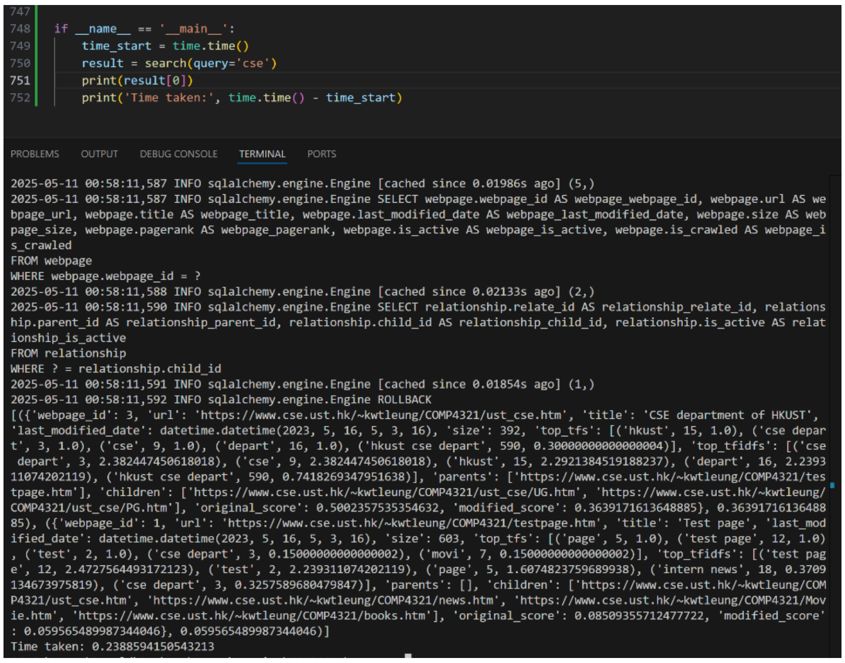
```
WHERE keyword.word_id = ?
2025-05-11 14:26:32,440 INFO sqlalchemy.engine.Engine [cached since 8.974s ago] (3128,)
2025-05-11 14:26:32,441 INFO sqlalchemy.engine.Engine SELECT keyword.word_id AS keyword_word_id, keyword.word AS keyword_word
FROM keyword
WHERE keyword.word_id = ?
2025-05-11 14:26:32,441 INFO sqlalchemy.engine.Engine [cached since 8.975s ago] (3068,)
2025-05-11 14:26:32,442 INFO sqlalchemy.engine.Engine SELECT keyword.word_id AS keyword_word_id, keyword.word AS keyword_word
FROM keyword
WHERE keyword.word_id = ?
2025-05-11 14:26:32,443 INFO sqlalchemy.engine.Engine [cached since 8.976s ago] (3085,)
2025-05-11 14:26:32,444 INFO sqlalchemy.engine.Engine SELECT relationship.relate_id AS relationship_relate_id, relationship.parent_id AS relat
ionship_parent_id, relationship.child_id AS relationship_child_id, relationship.is_active AS relationship_is_active
FROM relationship
WHERE ? = relationship.parent_id
2025-05-11 14:26:32,445 INFO sqlalchemy.engine.Engine [cached since 11.9s ago] (297,)
2025-05-11 14:26:32,447 INFO sqlalchemy.engine.Engine SELECT relationship.relate_id AS relationship_relate_id, relationship.parent_id AS relat
ionship_parent_id, relationship.child_id AS relationship_child_id, relationship.is_active AS relationship_is_active
FROM relationship
WHERE ? = relationship.child_id
2025-05-11 14:26:32,447 INFO sqlalchemy.engine.Engine [cached since 12s ago] (297,)
2025-05-11 14:26:32,609 INFO sqlalchemy.engine.Engine ROLLBACK
Time taken: 113.13078212738037
```



```
Page 1
Title: Test page
URL: https://www.cse.ust.hk/~kwtleung/COMP4321/testpage.htm
Last Modified Date: 2023-05-16 05:03:16
Page Size: 603
Title Keywords:
    - "page" with frequency: 1
    - "test page" with frequency: 1
    - "test" with frequency: 1
Body Keywords:
    - "cse depart" with frequency: 1
    - "movi" with frequency: 1
    - "intern news" with frequency: 1
    - "hkust" with frequency: 1
    - "intern" with frequency: 1
    - "movi list" with frequency: 1
    - "get" with frequency: 1
    - "depart" with frequency: 1
    - "list" with frequency: 1
    - "test page" with frequency: 2
Children:
    - https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse.htm
    - https://www.cse.ust.hk/~kwtleung/COMP4321/news.htm
    - https://www.cse.ust.hk/~kwtleung/COMP4321/Movie.htm
    - https://www.cse.ust.hk/~kwtleung/COMP4321/books.htm
Parent:
    No Parent Found
----------------------------------
Page 2
Title: books
URL: https://www.cse.ust.hk/~kwtleung/COMP4321/books.htm
Last Modified Date: 2023-05-16 05:03:16
Page Size: 601
Title Keywords:
    - "book" with frequency: 1
Body Keywords:
```

Testing on Basic Search (search.py)

Simple searching with only query provided should take less than 20 seconds (0.2 seconds in this case for "cse" query) for searching. A JSON array of webpage details and corresponding scores is printed which will be passed to frontend for formatting.

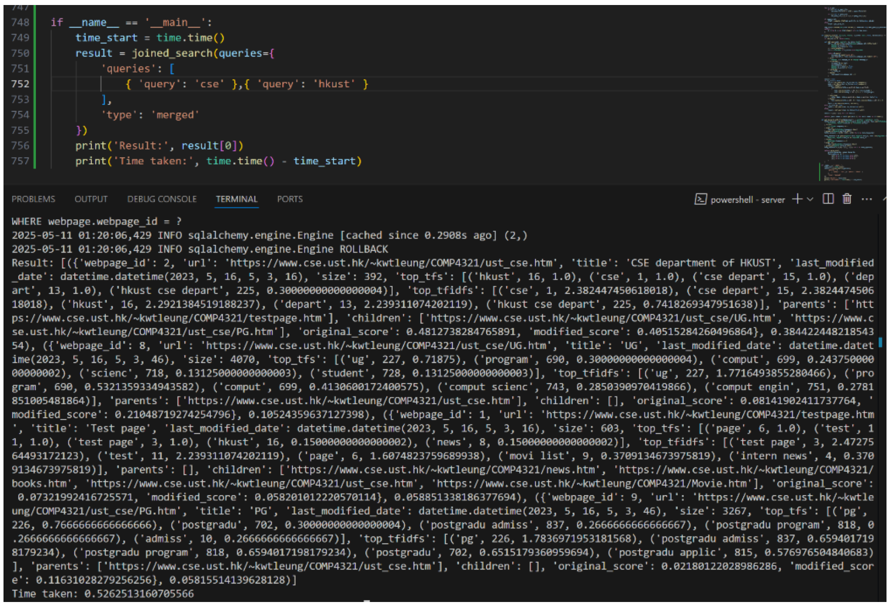


```python
if __name__ == '__main__':
    time_start = time.time()
    result = search(query='cse')
    print(result[0])
    print('Time taken:', time.time() - time_start)
```

```
2025-05-11 00:58:11,587 INFO sqlalchemy.engine.Engine [cached since 0.01986s ago] (5,)
2025-05-11 00:58:11,587 INFO sqlalchemy.engine.Engine SELECT webpage.webpage_id AS webpage_webpage_id, webpage.url AS we
bpage_url, webpage.title AS webpage_title, webpage.last_modified_date AS webpage_last_modified_date, webpage.size AS web
page_size, webpage.pagerank AS webpage_pagerank, webpage.is_active AS webpage_is_active, webpage.is_crawled AS webpage_i
s_crawled
FROM webpage
WHERE webpage.webpage_id = ?
2025-05-11 00:58:11,588 INFO sqlalchemy.engine.Engine [cached since 0.02133s ago] (2,)
2025-05-11 00:58:11,590 INFO sqlalchemy.engine.Engine SELECT relationship.relate_id AS relationship_relate_id, relations
hip.parent_id AS relationship_parent_id, relationship.child_id AS relationship_child_id, relationship.is_active AS relat
ionship_is_active
FROM relationship
WHERE ? = relationship.child_id
2025-05-11 00:58:11,591 INFO sqlalchemy.engine.Engine [cached since 0.01854s ago] (1,)
2025-05-11 00:58:11,592 INFO sqlalchemy.engine.Engine ROLLBACK
[({'webpage_id': 3, 'url': 'https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse.htm', 'title': 'CSE department of HKUST',
'last_modified_date': datetime.datetime(2023, 5, 16, 5, 3, 16), 'size': 392, 'top_tfs': [('hkust', 15, 1.0), ('cse depar
t', 3, 1.0), ('cse', 9, 1.0), ('depart', 16, 1.0), ('hkust cse depart', 590, 0.30000000000000004)], 'top_tfidfs': [('cse
 depart', 3, 2.382447450618018), ('cse', 9, 2.382447450618018), ('hkust', 15, 2.2921384519188237), ('depart', 16, 2.2393
11074202119), ('hkust cse depart', 590, 0.7418269347951638)], 'parents': ['https://www.cse.ust.hk/~kwtleung/COMP4321/tes
tpage.htm'], 'children': ['https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse/UG.htm', 'https://www.cse.ust.hk/~kwtleung/
COMP4321/ust_cse/PG.htm'], 'original_score': 0.5002357535354632, 'modified_score': 0.3639171613648885}, 0.36391716136488
85), ({'webpage_id': 1, 'url': 'https://www.cse.ust.hk/~kwtleung/COMP4321/testpage.htm', 'title': 'Test page', 'last_mod
ified_date': datetime.datetime(2023, 5, 16, 5, 3, 16), 'size': 603, 'top_tfs': [('page', 5, 1.0), ('test page', 12, 1.0)
, ('test', 2, 1.0), ('cse depart', 3, 0.15000000000000002), ('movi', 7, 0.15000000000000002)], 'top_tfidfs': [('test pag
e', 12, 2.4727564493172123), ('test', 2, 2.239311074202119), ('page', 5, 1.6074823759689938), ('intern news', 18, 0.3709
134673975819), ('cse depart', 3, 0.3257589680479847)], 'parents': [], 'children': ['https://www.cse.ust.hk/~kwtleung/COM
P4321/ust_cse.htm', 'https://www.cse.ust.hk/~kwtleung/COMP4321/news.htm', 'https://www.cse.ust.hk/~kwtleung/COMP4321/Mov
ie.htm', 'https://www.cse.ust.hk/~kwtleung/COMP4321/books.htm'], 'original_score': 0.08509355712477722, 'modified_score'
: 0.059565489987344046}, 0.059565489987344046)]
Time taken: 0.2388594150543213
```

Testing on Merged Search (search.py)

Joined searching (subquery and merged search) would have searching time proportional to the number of queries inputted (0.5 seconds in this case). A JSON array of webpage details and corresponding average scores is printed which will be passed to frontend for formatting.



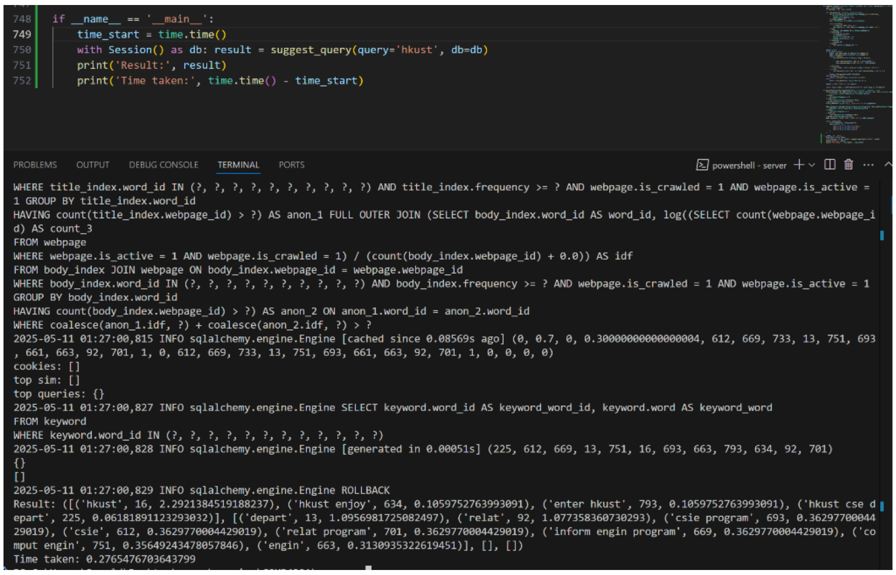
```python
if __name__ == '__main__':
    time_start = time.time()
    result = joined_search(queries={
        'queries': [
            { 'query': 'cse' },{ 'query': 'hkust' }
        ],
        'type': 'merged'
    })
    print('Result:', result[0])
    print('Time taken:', time.time() - time_start)
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
WHERE webpage.webpage_id = ?
2025-05-11 01:20:06,429 INFO sqlalchemy.engine.Engine [cached since 0.2908s ago] (2,)
2025-05-11 01:20:06,429 INFO sqlalchemy.engine.Engine ROLLBACK
Result: [({'webpage_id': 2, 'url': 'https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse.htm', 'title': 'CSE department of HKUST', 'last_modified
_date': datetime.datetime(2023, 5, 16, 5, 3, 16), 'size': 392, 'top_tfs': [('hkust', 16, 1.0), ('cse', 1, 1.0), ('cse depart', 15, 1.0), ('dep
art', 13, 1.0), ('hkust cse depart', 225, 0.30000000000000004)], 'top_tfidfs': [('cse', 1, 2.382447450618018), ('cse depart', 15, 2.3824474506
18018), ('hkust', 16, 2.2921384519188237), ('depart', 13, 2.239311074202119), ('hkust cse depart', 225, 0.7418269347951638)], 'parents': ['htt
ps://www.cse.ust.hk/~kwtleung/COMP4321/testpage.htm'], 'children': ['https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse/UG.htm', 'https://www.c
se.ust.hk/~kwtleung/COMP4321/ust_cse/PG.htm'], 'original_score': 0.4812738284765891, 'modified_score': 0.40515284260496864}, 0.384422448218543
54), ({'webpage_id': 8, 'url': 'https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse/UG.htm', 'title': 'UG', 'last_modified_date': datetime.datet
ime(2023, 5, 16, 5, 3, 46), 'size': 4070, 'top_tfs': [('ug', 227, 0.71875), ('program', 690, 0.30000000000000004), ('comput', 699, 0.243750000
00000002), ('scienc', 718, 0.13125000000000003), ('student', 728, 0.13125000000000003)], 'top_tfidfs': [('ug', 227, 1.7716493855280466), ('pro
gram', 690, 0.5321359334943582), ('comput', 699, 0.4130600172400575), ('comput scienc', 743, 0.2850390970419866), ('comput engin', 751, 0.2781
851005481864)], 'parents': ['https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse.htm'], 'children': [], 'original_score': 0.08141902411737764, '
modified_score': 0.21048719274254796}, 0.10524359637127398), ({'webpage_id': 1, 'url': 'https://www.cse.ust.hk/~kwtleung/COMP4321/testpage.htm
', 'title': 'Test page', 'last_modified_date': datetime.datetime(2023, 5, 16, 5, 3, 16), 'size': 603, 'top_tfs': [('page', 6, 1.0), ('test', 1
1, 1.0), ('test page', 3, 1.0), ('hkust', 16, 0.15000000000000002), ('news', 8, 0.15000000000000002)], 'top_tfidfs': [('test page', 3, 2.47275
64493172123), ('test', 11, 2.239311074202119), ('page', 6, 1.6074823759689938), ('movi list', 9, 0.3709134673975819), ('intern news', 4, 0.370
9134673975819)], 'parents': [], 'children': ['https://www.cse.ust.hk/~kwtleung/COMP4321/news.htm', 'https://www.cse.ust.hk/~kwtleung/COMP4321/
books.htm', 'https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse.htm', 'https://www.cse.ust.hk/~kwtleung/COMP4321/Movie.htm'], 'original_score':
 0.07321992416725571, 'modified_score': 0.058201012220570114}, 0.058851338186377694), ({'webpage_id': 9, 'url': 'https://www.cse.ust.hk/~kwtle
ung/COMP4321/ust_cse/PG.htm', 'title': 'PG', 'last_modified_date': datetime.datetime(2023, 5, 16, 5, 3, 46), 'size': 3267, 'top_tfs': [('pg',
226, 0.7666666666666666), ('postgradu', 702, 0.30000000000000004), ('postgradu admiss', 837, 0.26666666666666667), ('postgradu program', 818, 0
.26666666666666667), ('admiss', 10, 0.26666666666666667)], 'top_tfidfs': [('pg', 226, 1.7836971953181568), ('postgradu admiss', 837, 0.659401719
8179234), ('postgradu program', 818, 0.6594017198179234), ('postgradu', 702, 0.6515179360959694), ('postgradu applic', 815, 0.576976504840683)
], 'parents': ['https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse.htm'], 'children': [], 'original_score': 0.02180122028986286, 'modified_scor
e': 0.11631028279256256}, 0.05815514139628128)]
Time taken: 0.5262513160705566
```

Testing on Query Suggestion (search.py)

Query suggestion should take less than 10 seconds to process (0.2 seconds in this case for suggesting query for “hkust”). A JSON object of query details is outputted which will be passed to frontend.



```python
if __name__ == '__main__':
    time_start = time.time()
    with Session() as db: result = suggest_query(query='hkust', db=db)
    print('Result:', result)
    print('Time taken:', time.time() - time_start)
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
WHERE title_index.word_id IN (?, ?, ?, ?, ?, ?, ?, ?, ?, ?) AND title_index.frequency >= ? AND webpage.is_crawled = 1 AND webpage.is_active =
1 GROUP BY title_index.word_id
HAVING count(title_index.webpage_id) > ?) AS anon_1 FULL OUTER JOIN (SELECT body_index.word_id AS word_id, log((SELECT count(webpage.webpage_i
d) AS count_3
FROM webpage
WHERE webpage.is_active = 1 AND webpage.is_crawled = 1) / (count(body_index.webpage_id) + 0.0)) AS idf
FROM body_index JOIN webpage ON body_index.webpage_id = webpage.webpage_id
WHERE body_index.word_id IN (?, ?, ?, ?, ?, ?, ?, ?, ?, ?) AND body_index.frequency >= ? AND webpage.is_crawled = 1 AND webpage.is_active = 1
GROUP BY body_index.word_id
HAVING count(body_index.webpage_id) > ?) AS anon_2 ON anon_1.word_id = anon_2.word_id
WHERE coalesce(anon_1.idf, ?) + coalesce(anon_2.idf, ?) > ?
2025-05-11 01:27:00,815 INFO sqlalchemy.engine.Engine [cached since 0.08569s ago] (0, 0.7, 0, 0.30000000000000004, 612, 669, 733, 13, 751, 693
, 661, 663, 92, 701, 1, 0, 612, 669, 733, 13, 751, 693, 661, 663, 92, 701, 1, 0, 0, 0, 0)
cookies: []
top sim: []
top queries: {}
2025-05-11 01:27:00,827 INFO sqlalchemy.engine.Engine SELECT keyword.word_id AS keyword_word_id, keyword.word AS keyword_word
FROM keyword
WHERE keyword.word_id IN (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
2025-05-11 01:27:00,828 INFO sqlalchemy.engine.Engine [generated in 0.00051s] (225, 612, 669, 13, 751, 16, 693, 663, 793, 634, 92, 701)
{}
[]
2025-05-11 01:27:00,829 INFO sqlalchemy.engine.Engine ROLLBACK
Result: ([('hkust', 16, 2.2921384519188237), ('hkust enjoy', 634, 0.1059752763993091), ('enter hkust', 793, 0.1059752763993091), ('hkust cse d
epart', 225, 0.06181891123293032)], [('depart', 13, 1.0956981725082497), ('relat', 92, 1.077358360730293), ('csie program', 693, 0.36297700044
29019), ('csie', 612, 0.3629770004429019), ('relat program', 701, 0.3629770004429019), ('inform engin program', 669, 0.3629770004429019), ('co
mput engin', 751, 0.35649243478057846), ('engin', 663, 0.3130935322619451)], [], [])
Time taken: 0.2765476703643799
```

7. Conclusion

Strengths:

- Comprehensive implementation of core features (crawler, indexer, search engine).
- Effective use of bonus features (PageRank, PMI, relevance feedback) to enhance usability.
- Modular design with clear database schemas and algorithm documentation.

Weaknesses:

- Dependency on external libraries (e.g., NLTK) may affect reproducibility.
- SQLite may limit scalability compared to distributed databases.

Future Improvements:

- Implement distributed crawling for larger datasets.
- Add caching mechanisms to improve search speed.
- Support real-time indexing for dynamic content.

8. Contribution

- Cheung Tuen King: 40% (Mainly responsible for server-side and client-side development)
- Li Ching Ho: 30% (Mainly responsible for video recording)
- Chan Yuk Ki: 30% (Mainly responsible for documentation)