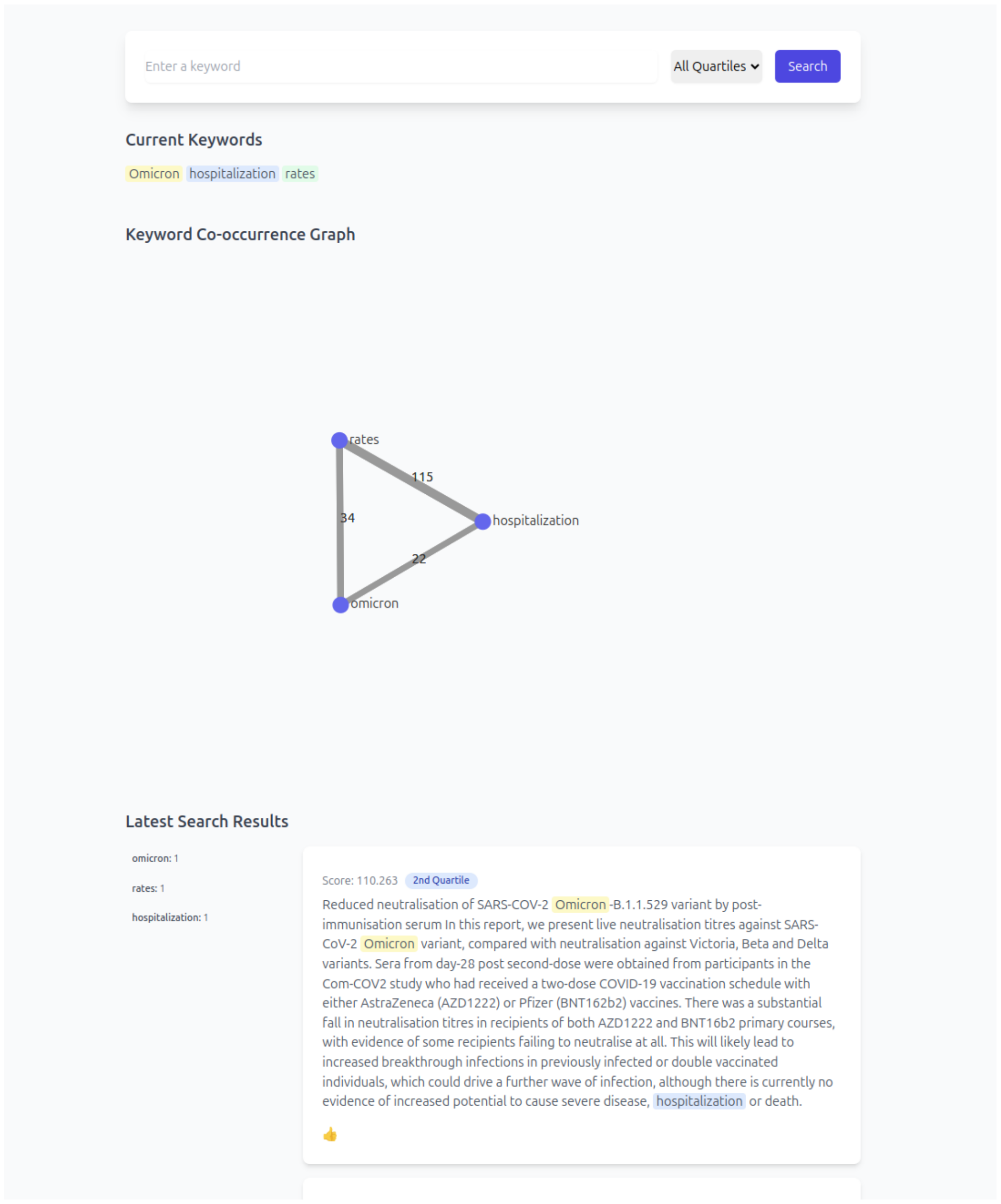
# Medical Article Discovery - Documentation

This project is a Flask-based application that allows users to search, highlight keywords, and explore text data.



Reduced neutralisation of SARS-COV-2 Omicron -B.1.1.529 variant by post-immunisation serum In this report, we present live neutralisation titres against SARS-CoV-2 Omicron variant, compared with neutralisation against Victoria, Beta and Delta variants. Sera from day-28 post second-dose were obtained from participants in the Com-COV2 study who had received a two-dose COVID-19 vaccination schedule with either AstraZeneca (AZD1222) or Pfizer (BNT162b2) vaccines. There was a substantial fall in neutralisation titres in recipients of both AZD1222 and BNT16b2 primary courses, with evidence of some recipients failing to neutralise at all. This will likely lead to increased breakthrough infections in previously infected or double vaccinated individuals, which could drive a further wave of infection, although there is currently no evidence of increased potential to cause severe disease, hospitalization or death.

Figure 1: **Screenshot of the application.** The user can enter keywords and search for articles. The search results are displayed with the keywords highlighted. A keyword co-occurrence graph is also displayed.

## Contents

# 1 Main Functions

## 1.1 app.py

- **highlight_keywords**: Located in app.py as `highlight_keywords`. It highlights specified terms in the displayed text.
- **start_search**: Defined in app.py as `start_search`. Initializes a new search session and redirects to the main search page.
- **search_page**: Located in app.py as `search_page`. Displays the search page with the search results. Calculates the BM25 score for the search query, co-occurrence of keywords, and entropy-based rankings.
- **click_document**: Defined in app.py as `click_document`. Records a click on a document and updates the click count.

## 1.2 data_loader.py

- **load_initial_data**: Located in data_loader.py as `load_initial_data`. Loads initial data for the application.
- **load_dataset**: Defined in data_loader.py as `load_dataset`. Loads the dataset from a JSON file.
- **process_dataset**: Located in data_loader.py as `process_dataset`. Processes the samples in the dataset.

## 1.3 search.py

- **bm25_search**: Located in search.py as `bm25_search`. Executes a BM25 search using the raw SQL from queries.yaml.
- **get_results_texts**: Defined in search.py as `get_results_texts`. Given a list of results, returns the full text for each result.
- **get_doc_entropy_quartiles**: Located in search.py as `get_doc_entropy_quartiles`. Given a list of document IDs, returns the entropy quartile for each document.
- **get_keyword_cooccurrences**: Defined in search.py as `get_keyword_cooccurrences`. Retrieves the co-occurrence of keywords in the dataset.
- **get_keyword_clickthroughs**: Located in search.py as `get_keyword_clickthroughs`. Retrieves the clickthrough rate for keywords in the dataset.

# 2 Configuration

The configuration file config.yaml contains the configuration settings for the application. The file specifies the database connection details, and the search parameters.

- **DB_HOST**: The hostname of the database server. Default is `localhost`.
- **DB_USER**: The username to connect to the database. Default is `root`.
- **DB_PASSWORD**: The password to connect to the database. Default is `5891326`.
- **DB_NAME**: The name of the database. Default is `db06`.
- **K1**: The BM25 parameter that controls term frequency saturation. Default is `1.2`.
- **B**: The BM25 parameter that controls document length normalization. Default is `0.75`.

## 3 SQL Queries

The file queries.yaml contains queries for co-occurrence of keywords and to retrieve entropy-based rankings.

### 3.1 tfidf_entropy

This query calculates the entropy of documents based on their TF-IDF scores. It is used to rank documents based on their relevance to a search query.

```
WITH doc_tfidf AS (
SELECT
    dv.document_id AS doc_id,
    (dv.term_frequency / d.doc_length) * df.idf AS tfidf
FROM document_vocabulary dv
JOIN documents d ON d.id = dv.document_id
JOIN document_frequencies df ON df.vocabulary_id = dv.vocabulary_id
WHERE dv.document_id IN (:doc_ids)
),
doc_sums AS (
SELECT doc_id, SUM(tfidf) AS total_tfidf
FROM doc_tfidf
GROUP BY doc_id
),
doc_probs AS (
SELECT
    t.doc_id,
    (t.tfidf / s.total_tfidf) AS p
FROM doc_tfidf t
JOIN doc_sums s ON t.doc_id = s.doc_id
),
doc_entropy AS (
SELECT
    doc_id,
    -1 * SUM(p * LN(p)) AS entropy
FROM doc_probs
WHERE p > 0
GROUP BY doc_id
),
ranked_docs AS (
SELECT
    doc_id,
    entropy,
    CUME_DIST() OVER (ORDER BY entropy) AS percentile
FROM doc_entropy
)
SELECT
doc_id,
entropy,
percentile * 100 AS percentile_score,
CASE
    WHEN percentile <= 0.25 THEN '1st-Quartile'
    WHEN percentile <= 0.50 THEN '2nd-Quartile'
    WHEN percentile <= 0.75 THEN '3rd-Quartile'
    ELSE '4th-Quartile'
END AS entropy_quartile
FROM ranked_docs
ORDER BY entropy;
```

### 3.2 get_keyword_cooccurrences

This query retrieves the co-occurrence of keywords in the dataset.

```
SELECT
v1.term AS kw1,
v2.term AS kw2,
COUNT(DISTINCT dv1.document_id) AS cooccurrence_count
FROM document_vocabulary dv1
JOIN document_vocabulary dv2
ON dv1.document_id = dv2.document_id
JOIN vocabulary v1
ON dv1.vocabulary_id = v1.id
JOIN vocabulary v2
ON dv2.vocabulary_id = v2.id
WHERE dv1.vocabulary_id != dv2.vocabulary_id
AND v1.term IN (:terms)
AND v2.term IN (:terms)
GROUP BY
v1.term, v2.term;
```

### 3.3 search_bm25

This query calculates the BM25 score for documents based on a search query.

```
SELECT
d.id AS doc_id,
SUM(
    df.idf * (
        (dv.term_frequency * ( :k1 + 1 ))
        / ( dv.term_frequency + :k1 * (1 - :b + :b * (d.doc_length / :avgdl)) )
    )
) AS bm25_score
FROM documents d
JOIN document_vocabulary dv ON d.id = dv.document_id
JOIN vocabulary v ON dv.vocabulary_id = v.id
JOIN document_frequencies df ON df.vocabulary_id = v.id
WHERE v.term IN (:query_terms)
GROUP BY d.id
ORDER BY bm25_score DESC
LIMIT :limit;
```

## 4 Datasets

Datasets are stored in the `/datasets` folder:

- `biorxiv.jsonl` - Contains articles from the bioRxiv preprint server. Collected for the MTEB project.
- `medrxiv.jsonl` - Contains articles from the MedrXiv preprint server. Collected for the MTEB project.

These files contain the text corpus used by the application.

## 5 Database Schema

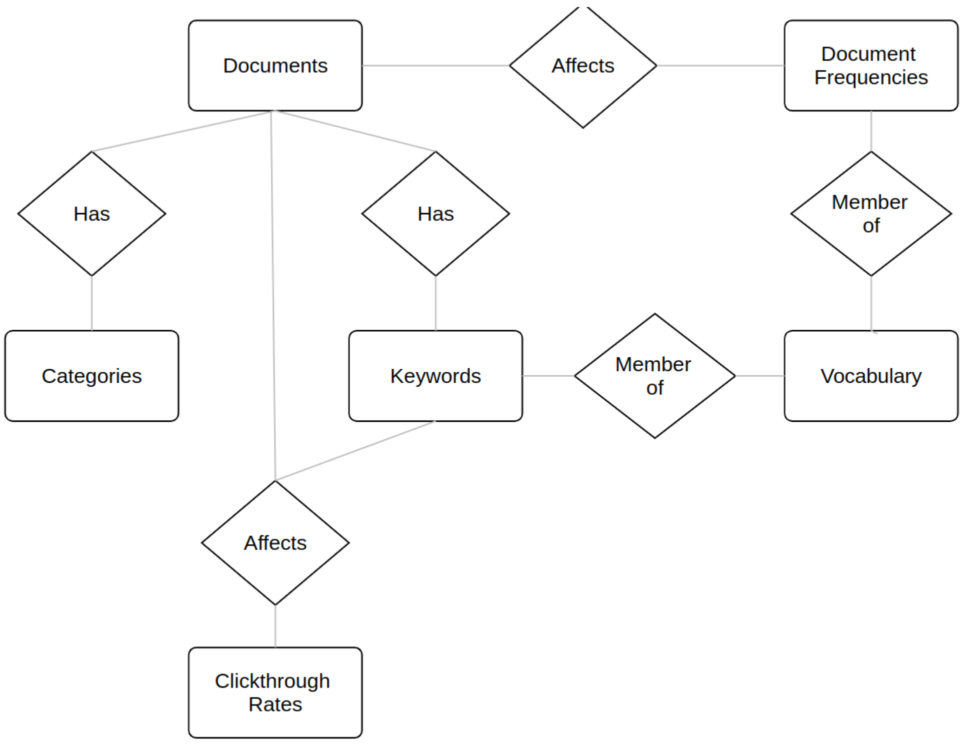


Figure 2: **Entity-Relationship Diagram (ERD) of the database schema.**

### 5.1 categories

This table stores the categories of the documents.

- **id**: INT, Primary Key, Auto Increment
- **name**: VARCHAR(255), Not Null

### 5.2 datasets

This table stores the dataset names used in the application.

- **id**: INT, Primary Key, Auto Increment
- **name**: VARCHAR(255), Not Null

### 5.3 documents

This table stores the documents in the database.

- **id**: INT, Primary Key, Auto Increment
- **abstract**: TEXT, Not Null
- **category_id**: INT, Not Null, Foreign Key (references `categories(id)`)
- **doc_length**: INT, Not Null
- **dataset_id**: INT, Not Null, Foreign Key (references `datasets(id)`)

### 5.4 vocabulary

This table stores the vocabulary terms.

- **id**: INT, Primary Key, Auto Increment
- **term**: VARCHAR(255), Not Null

### 5.5 document_frequencies

This table stores the document frequencies for the vocabulary terms.

- **id**: INT, Primary Key, Auto Increment
- **vocabulary_id**: INT, Not Null, Foreign Key (references `vocabulary(id)`)
- **doc_freq**: INT, Not Null
- **idf**: DOUBLE, Not Null

### 5.6 keywords

This table stores the keywords extracted from the documents.

- **id**: INT, Primary Key, Auto Increment
- **keyword**: VARCHAR(255), Not Null
- **vocabulary_id**: INT, Null, Foreign Key (references `vocabulary(id)`)

### 5.7 document_keywords

This table stores the keywords associated with each document.

- **document_id**: INT, Not Null, Foreign Key (references `documents(id)`)
- **keyword_id**: INT, Not Null, Foreign Key (references `keywords(id)`)
- **term_frequency**: INT, Not Null
- **Primary Key**: (document_id, keyword_id)

### 5.8 document_vocabulary

This table stores the vocabulary terms associated with each document.

- **document_id**: INT, Not Null, Foreign Key (references `documents(id)`)
- **vocabulary_id**: INT, Not Null, Foreign Key (references `vocabulary(id)`)
- **term_frequency**: INT, Not Null
- **Primary Key**: (document_id, vocabulary_id)

### 5.9 clickthrough_rates

This table stores the clickthrough rates for the keywords.

- **id**: INT, Primary Key, Auto Increment
- **document_id**: INT, Not Null, Foreign Key (references `documents(id)`)
- **term_id**: INT, Not Null, Foreign Key (references `vocabulary(id)`)
- **clicks**: INT, Not Null, Default 0
- **Unique**: (document_id, term_id)

## 6 How to Use the App

1. (If needed) Run the build script `Build.sh` to set up dependencies, download the original datasets, create a virtual environment, and initialize the database.
2. Start the Flask server by running:

   ```
   python app.py
   ```

   At first run, if the database is not populated, the application will load the dataset, process them and populate the database. This may take some time.
3. Open your browser at `http://127.0.0.1:5000/`.
4. Enter keywords and launch searches. The keywords will be highlighted in the results.
5. Explore the text data and use the search functionality.