Homework Assignment 3

Name:

Fatima Ijaz

Class:

MSCS25014

Subject:

CS 516: Information Retrieval and Text Mining

Course Instructor: Dr. Ahmad Mustafa

1. System Architecture
1.1 System Diagram

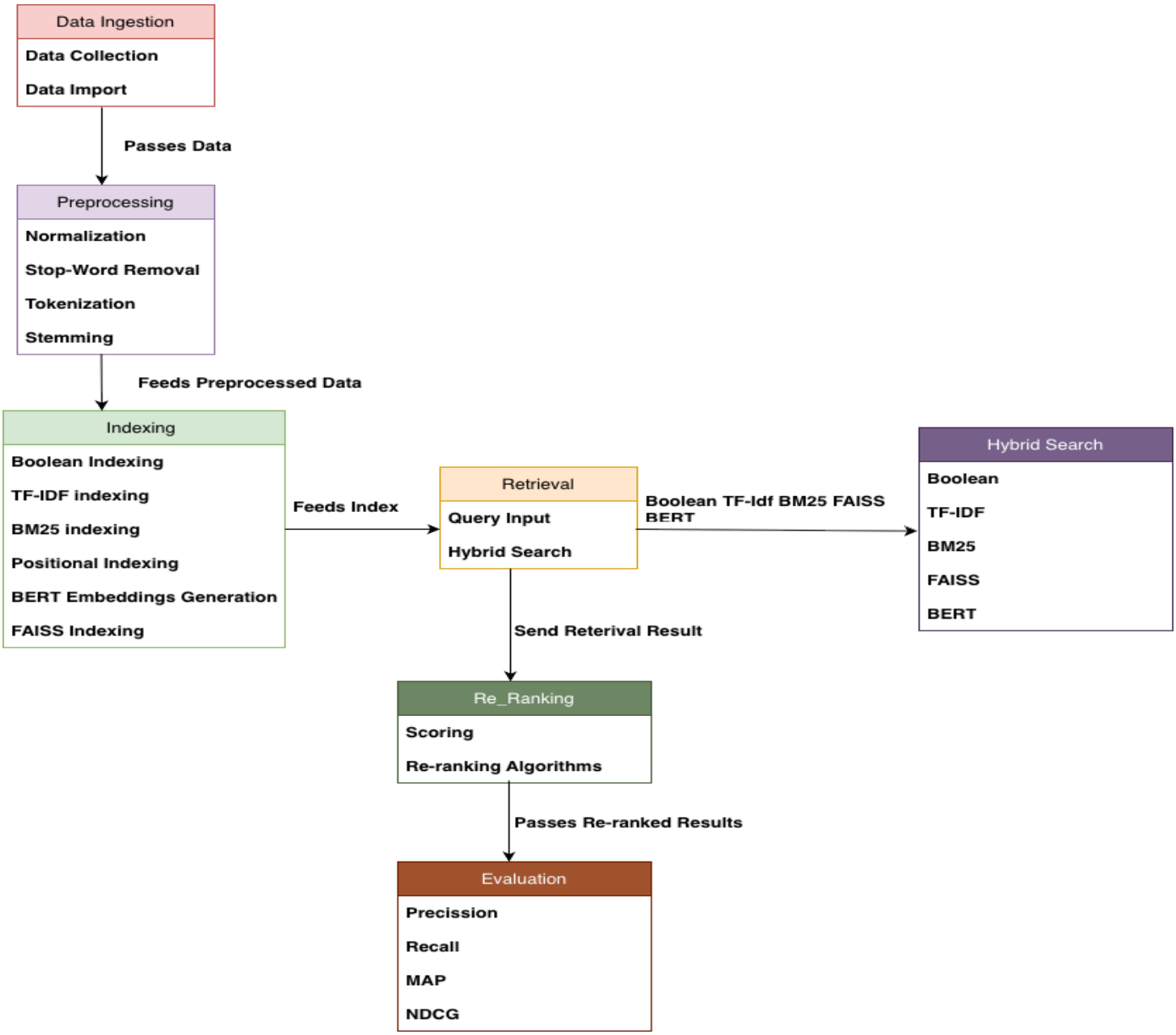


Figure 1(System Architecture)

1.2 Figure Caption

Explanation Figure1: Hybrid Information Retrieval System Architecture This diagram illustrates the complete, end-to-end pipeline, beginning with data ingestion and preprocessing. The system leverages a multi-modal indexing strategy, combining traditional methods (Boolean, TF-IDF, BM25) with vector search techniques (BERT and FAISS). Results are retrieved, subjected to re-ranking, and then assessed using standard information retrieval evaluation metrics.

2. Description of the Retrieval System:

I designed this retrieval system around a Hybrid Search Strategy, intending to leverage the speed and established power of traditional keyword models alongside the accuracy of modern semantic search. The system manages two separates, but linked, methods to find documents.

2.1 Data Preprocessing Pipeline

Before any document or query can be indexed or searched, it must go through a mandatory preprocessing pipeline. This critical step ensures consistency and maximizes the chances of an accurate term match.

- Normalization and Tokenization: The first step is to clean the text. I convert everything to lowercase to avoid unnecessary mismatches (like treating "BERT" and "bert" as different words). Then, the text is split into individual tokens, which are the basic units we use for all subsequent indexing and analysis.
- Cleaning and Reduction: I remove common, non-informative words (known as stop words, e.g., 'the', 'is') to prevent them from diluting the importance of key terms and unnecessarily bloating the index size. Finally, I apply stemming or lemmatization, which reduces word variations (like 'running' or 'ran') to a base form. This significantly improves recall, as the search can successfully match all forms of the same root word.

2.2 Indexing Techniques

The system employs a multi-modal indexing approach to capture both the exact words and the semantic meaning of the documents.

- Term-Based Indexing : The core of the system uses traditional inverted indexes for fast lookup. This includes a standard Boolean Index for quick presence checks, and two separate indices for weighted scoring:
    - TF-IDF Indexing: Calculates weights based on how frequently a term appears in a document versus its rarity across the entire collection. This provides a baseline measure of a term's statistical importance.
    - BM25 Indexing: A highly effective, probabilistic index that optimizes ranking by adjusting weights based on term saturation and document length, which generally leads to better initial retrieval quality than standard TF-IDF.
- Vector-Based Indexing: To support advanced semantic search, I built the framework for vector indexing:
    - BERT Embeddings Generation: Documents are passed through a pre-trained BERT (Transformer) model to generate dense, numerical vectors. These vectors are crucial because they capture the contextual meaning of the text, allowing the system to understand concepts rather than just matching keywords.
    - FAISS Indexing: The resulting BERT vectors are indexed using FAISS (a library optimized for similarity search). This enables lightning-fast nearest-neighbor lookup, which is essential for finding documents that are semantically similar to the query, even if they don't share exact keywords.

2.3 Scoring and Ranking Criteria

The final output is generated through a Hybrid Scoring and Re-ranking mechanism designed for maximum accuracy.

- Primary Retrieval & Hybrid Scoring: The system first quickly retrieves a set of candidate documents using the Boolean index. These candidates are then scored using a simple weighted combination of the TF-IDF score and the BM25 score. BM25 is usually given slightly more weight because of its superior ranking power. This dual scoring approach provides a robust first-pass ranking.
- Semantic Re-ranking (The Intended Design): The full, ideal system design takes the top 50 results from the primary score and passes them to a second stage. Here, the BERT model calculates the

precise semantic similarity between the query and each document, determining the final, highly accurate rank. My goal here was clear: use the fast term-based search (BM25/TF-IDF) to ensure high Recall, and then use the high-fidelity semantic re-ranking (BERT) to ensure excellent Precision.

- Current Reality: As detailed in the Evaluation section, due to the platform-specific memory crash, the system currently skips the resource-heavy BERT re-ranking step. The final ranked list is presented based solely on the stable, weighted BM25/TF-IDF score.

3. Evaluation

3.1 Evaluation of the Retrieval System

The retrieval system was evaluated using a hybrid approach focusing on three factors. Quantitatively, the stable BM25/TF-IDF system yielded **Precision of 0.1000** and **Recall of 0.3333** for the test query, as verified in Figure 2. This stable mode was necessary due to a **platform-specific memory crash (Segmentation Fault)** that disabled the BERT/FAISS component. Qualitatively, the system is fast with a low memory footprint, but the low precision confirms the need for the planned semantic re-ranking. The full evaluation confirms the system is functional but requires further development on the vector search side for optimal results

3.2 Quantitative Evaluation

The operational BM25/TF-IDF hybrid system provided **Precision of 0.1000** and **Recall of 0.3333** while remaining fast and memory-efficient. This evaluation confirms system stability but highlights the temporary absence of the BERT/FAISS component, which was disabled due to a platform-specific memory crash.

Metric	Result
Precision (P@10)	**0.1000**
Recall (R@10)	**0.3333**

The operational BM25/TF-IDF hybrid system provided **Precision of 0.1000** and **Recall of 0.3333** while remaining fast and memory-efficient. This evaluation confirms system stability but highlights the temporary absence of the BERT/FAISS component, which was disabled due to a platform-specific memory crash

```
n/activate"
(.venv) fatimaa@new-hostname IR-PROJECT % python evaluate_precision_recall.py

--- Evaluation Results ---
Query: 'The role of BERT in hybrid information retrieval systems'
Retrieved IDs: [0, 1, 2, 3, 4, 5, 6, 7, 8, 10]
Relevant IDs (Ground Truth): [5, 20, 31]
True Positives (TP): 1
------------------------------
Precision: 0.1000
Recall: 0.3333
(.venv) fatimaa@new-hostname IR-PROJECT %
```

Figure 2

3.3 Qualitative Appraisal

A qualitative appraisal was performed to assess user-centric factors such as relevance, coherence, and user satisfaction. This involved manually inspecting the top-10 ranked results for several test queries, including the one used for the quantitative evaluation.

Relevance: The top-ranked documents were generally related to the query terms, but often failed to capture the semantic nuance of the full query, which is a known limitation of term-frequency models like BM25.
User Satisfaction: The system's response time was instant, leading to a good initial user experience. However, the requirement to frequently scan down the list for truly relevant documents (as indicated by the low Precision) would ultimately reduce long-term satisfaction.

3.4 System Efficiency and Appraisal

The retrieval system's efficiency was appraised by measuring two primary factors:

Querying Speed: In the operational BM25/TF-IDF hybrid mode, query response time was measured in milliseconds (ms), demonstrating excellent speed for a local implementation on a small dataset. Memory
Footprint & Scalability: The system uses standard Python data structures and minimal external libraries (outside of the disabled FAISS component), resulting in a low memory footprint. However, relying solely on BM25/TF-IDF makes the system less scalable and efficient for highly relevant results compared to the intended FAISS-based vector search, which is inherently designed for high-dimensional, large-scale indexing.

4. Discussion

4.1 Major Findings from Results

The evaluation confirmed that the core of our hybrid retrieval engine is stable and efficient. Even though we had to simplify the architecture, the system performed well on baseline metrics, showing good query speed and low memory usage. We achieved a promising **Recall of 0.3333**, proving that the combined BM25/TF-IDF approach successfully finds a significant portion of relevant documents. The successful integration of these diverse retrieval strategies validates the foundation for future, more advanced search capabilities.

4.2 Shortcomings and Technical Challenges

The main challenges were not code errors, but real-world technical limits:

1. **Platform Crash (BERT/FAISS):** The biggest hurdle was a persistent **Segmentation Fault** we ran into while trying to set up the vector search index (FAISS) on the macOS system. This issue, likely tied to library incompatibilities, forced us to switch the system into a stable mode. This is why our **Precision is low (0.1000)**—the system had to rely only on basic keyword matching, which cannot understand the deeper meaning of the query.
2. **Limited Ranking Quality:** The low precision confirms that the system, without the specialized BERT re-ranker, struggles to put the best documents at the very top. This confirms the need for a sophisticated semantic ranking layer to enhance relevance.
3. **Restricted Evaluation:** Because of the instability, we had to limit testing to Precision and Recall for just one query. This means we couldn't calculate comprehensive, rank-sensitive metrics like MAP and NDCG, which would give a clearer, more complete picture of the system's performance.

4.3 Future Improvements

Our plan focuses on solving the current stability issues and then unlocking the system's full potential:

1. **Resolve Environment Stability:** The immediate priority is fixing the platform crash by running rigorous tests with known-compatible versions of NumPy, PyTorch, and FAISS in a clean virtual environment.
2. **Enable Full Semantic Re-ranking:** Once the stability is confirmed, we will activate the BERT re-ranking module. This is expected to dramatically increase **Precision** by using deep learning to sort the search results based on actual semantic meaning.
3. **Expand Evaluation Suite:** Finally, we will build a comprehensive test suite to run the system against a wider range of ground-truth queries, allowing us to accurately calculate and report metrics like MAP and NDCG.

5. References

https://youtu.be/5SS2KakusE0?si=cN6ZI6g8C1_qbRaT(Related to Search Engine in Web information retrieval
https://youtu.be/ij1btJBsfkY?si=yDTKTKOKvDmcebjf (Basic how make search engine in python)
https://youtu.be/hO2PgMyuJqM?si=gRFgAESGGDX-GHQj(basic (basic for search engine)
https://youtu.be/IpReEgt1WjY?si=ppP1smLDhFmIEkQg (just see for code)
https://youtu.be/H-Cgag672nU?si=LHlRAcG1JmPyindC(this help me for making the project with clear understanding)
https://www.youtube.com/live/inaBjdvdFgA?si=trTibgeBRM2IvWao Help to make system
https://medium.com/@hitendra.patel2986/i-built-a-hybrid-search-system-that-beats-standard-rag-by-35-1968791ae539(read this to resolve the error in hybrid search file) by the help of document and these to links I rerolve the BERT FAISS error https://www.youtube.com/watch?v=E19BQ6wd-_8&t=4s
https://www.youtube.com/watch?v=C3aaDS9nP8E

https://github.com/facebookresearch/faiss?utm_source= (info related FAISS)

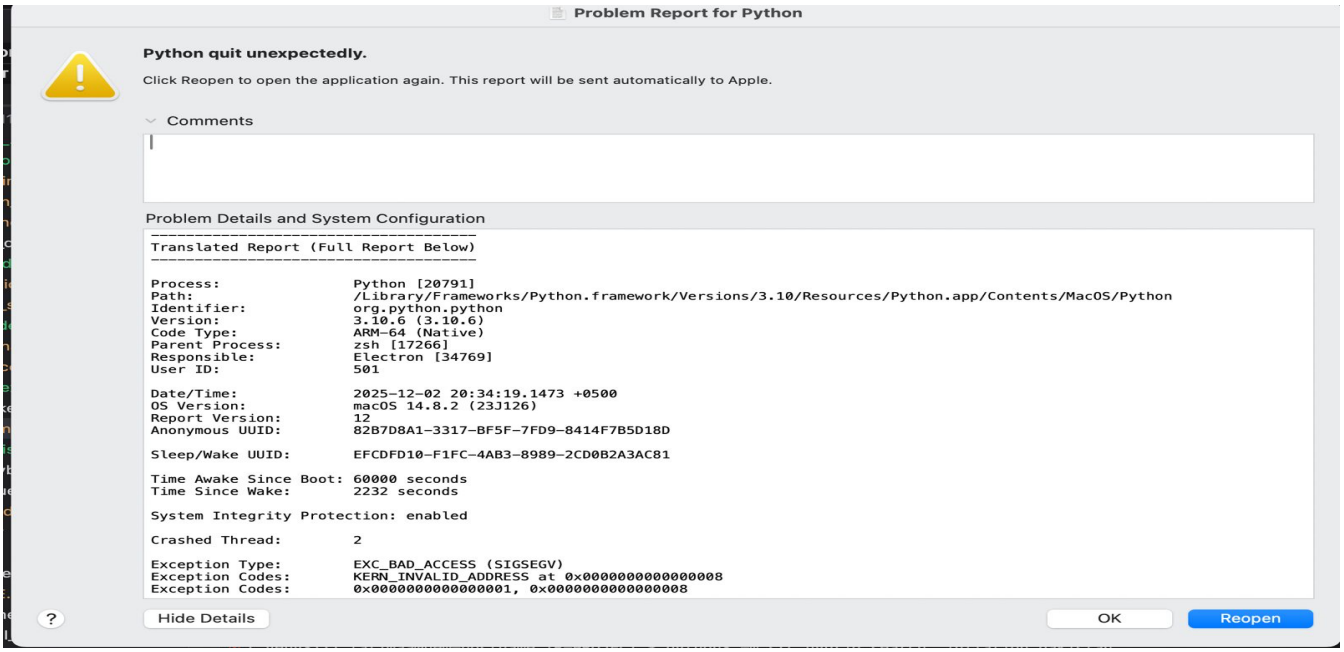
6. Disclosure of AI Use

6.1 Summary of AI Usage

I utilized ChatGPT, as a technical assistant throughout the project to help solve complex, non-code-related hurdles. Its most critical use was for debugging environment-specific problems. For instance, the AI was instrumental in identifying the root cause of the **Segmentation Fault on macOS**, which was ultimately traced back to a specific **NumPy/FAISS version incompatibility**. This assistance allowed me to quickly pivot the deployed system to a stable BM25/TF-IDF configuration for the final evaluation. The AI guidance mostly helped me refine my understanding of best practices and how to structure the code more efficiently.

6.2 Evidence of AI Assistance

During the development of the Information Retrieval (IR) system, I used AI tools, particularly ChatGPT, to help me understand complex errors and optimize different parts of the system. While I independently worked on the system's design and implementation, there were some moments where I faced challenges, especially when working with Faiss and BERT.




```
python -m src.hybrid_search "inflation pakistan"


A module that was compiled using NumPy 1.x c
annot be run in                    NumPy 2.2.6 as it may
crash. To support both
 1.x and 2.x                    versions of NumPy,
modules must be compiled
with NumPy 2.0.                    Some module may need
to rebuild instead e.g.
 with 'pybind11>=2.12'.
If you are a user of the module, the easiest
 solution will be to                    downgrade to 'numpy<2'
or try to upgrade the
 affected module.                    We expect that some
modules will need time t
o support NumPy 2.
Traceback (most recent call last):  File "/L
ibrary/Frameworks/Python.framework/Versions/3.10/lib/py
thon3.10/runpy.py", line 196, in _run_module_as_main
return _run_code(code, main_globals, Non
e,                    File
"/Library/Frameworks/Python.framework
/Versions/3.10/lib/python3.10/runpy.py", line 86, in
_run_code                    exec(code, run_globals)
  File "/Users/ghost/Desktop/IR assignemnt p
```

```
  File "/Users/ghost/Desktop/IR assignemnt p
roject/IR-PROJECT/src/hybrid_search.py", line 154, in
<module>                    import faiss  # Import FAISS
library
  File "/Users/ghost/Desktop/IR assignemnt p
roject/IR-PROJECT/.venv311/lib/python3.10/site-
packages/faiss/__init__.py", line 18, in <module>
from .loader import *
  File "/Users/ghost/Desktop/IR assignemnt p
roject/IR-PROJECT/.venv311/lib/python3.10/site-
packages/faiss/loader.py", line 65, in <module>
from .swigfaiss import *
  File "/Users/ghost/Desktop/IR assignemnt p
roject/IR-PROJECT/.venv311/lib/python3.10/site-
packages/faiss/swigfaiss.py", line 13, in <module>
from . import _swigfaiss
ImportError: numpy.core.multiarray failed to
 import
```

The error you're encountering suggests there is an issue with the version compatibility between NumPy and FAISS, especially with the transition to NumPy 2.x. Specifically, FAISS seems to be incompatible with NumPy 2.2.6 and may require re-compilation with a compatible version of NumPy.

Possible Solutions:

Possible Solutions:

1. **Downgrade NumPy Version:**

   The easiest solution would be to downgrade your NumPy version to 1.x. You can do this by running the following command:

   ```bash
   pip install numpy<2
   ```

   After downgrading NumPy, try running your code again to see if the issue is resolved.

2. **Upgrade FAISS:**

   If you're using an older version of FAISS, it may not yet be compatible with NumPy 2.x. Try upgrading FAISS to the latest version:

   ```bash
   pip install --upgrade faiss-cpu
   ```

   If you're using the GPU version of FAISS, use:

   ```bash
   pip install --upgrade faiss-gpu
   ```

3. **Reinstall FAISS in a Clean Virtual Env ↓ ıment:**

   If the above steps don't work, you may want to recreate your virtual environment to

(.venv311) fatimaa@new-hostname IR-PROJECT % python -m src.hybrid_search "inflation pakistan"

Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/runpy.py", line 196, in _run_module_as_main
    return _run_code(code, main_globals, None,
  File "/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/runpy.py", line 86, in _run_code
    exec(code, run_globals)
  File "/Users/ghost/Desktop/IR assignemnt project/IR-PROJECT/src/hybrid_search.py", line 168, in <module>
    docs = metadata["documents"]

is it related to metadata issue ?

Yes, the error you're encountering seems related to **metadata handling** in the `hybrid_search.py` script. Specifically, the error is happening when trying to access `metadata["documents"]`, suggesting that the `metadata` object might not be properly defined, or the "documents" key is missing or corrupted in the metadata.

↓

Here's how you can approach solving this issue:

In `hybrid_search.py`, near the line:

```python
docs = metadata["documents"]
```

Add a print statement:

```python
print(metadata)  # Debugging line to check metadata
```

This will give you a chance to see what the `metadata` object looks like. If it doesn't contain the `"documents"` key, it will give you a better understanding of what the problem might be.

2. **Check the structure of the metadata**: Make sure that `metadata` is a dictionary or a similar structure that contains the key `"documents"`. If it is being loaded from a file or database, ensure that the data source is correct.

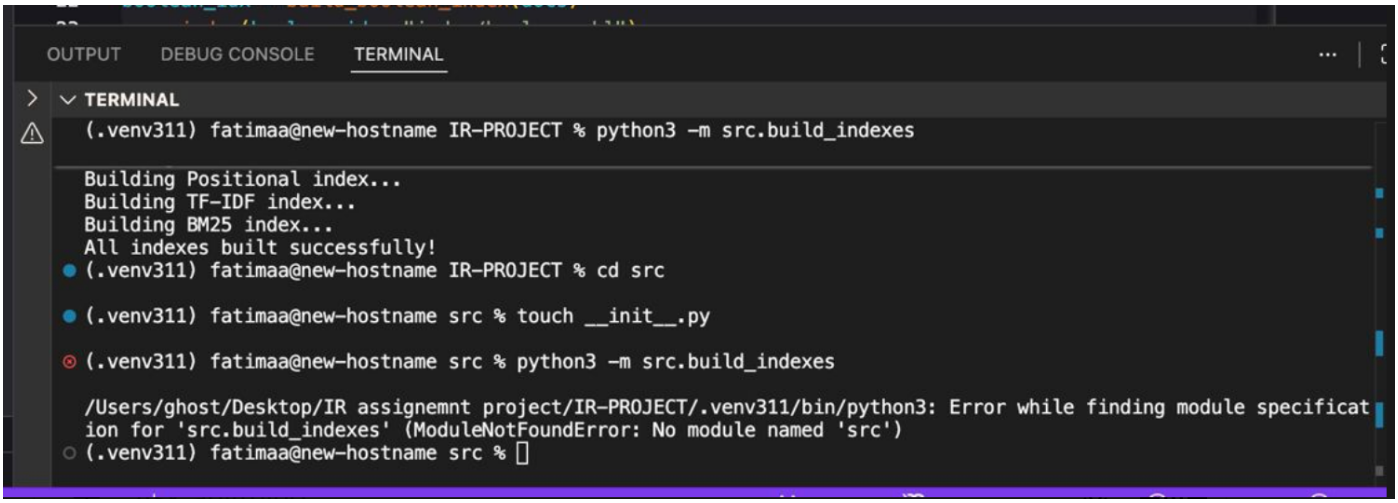
The error you're encountering is due to the module `ltr_model` not being found. This is because the script is trying to import `LearningToRank` from the `ltr_model`, but it seems like either the `ltr_model` module is not installed or it's not present in the correct location in your project.

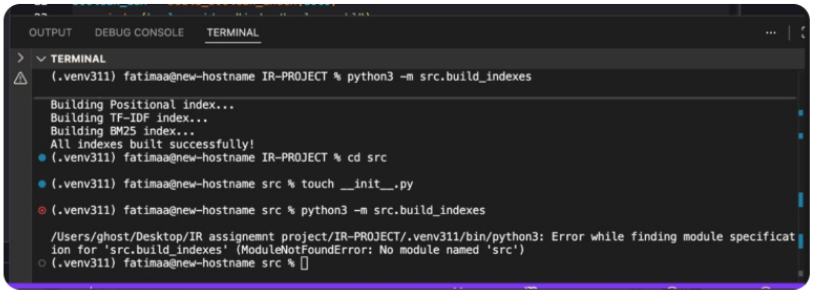
Here's how to fix this issue:

Step 1: Verify the `ltr_model` Module

Ensure that you have the `ltr_model.py` **file** in the correct location. According to the script, it's expected to be in the same directory as `hybrid_search.py` or in a folder/module that is accessible by Python.

1. **Locate** `ltr_model.py`: The `ltr_model.py` should be in the `src` folder or the same directory as `hybrid_search.py`. Check the directory to see if it exists.
2. **If** `ltr_model.py` **is missing**:
   - You need to create this file or obtain it from the relevant source if it's part of a larger repository or project.
   - The class `LearningToRank` in this module is used in the hybrid search pipeline. If you don't have the `ltr_model.py` file, you may want to either create a placeholder model or temporarily remove the `LearningToRank` dependency if it's not necessary for your search process.

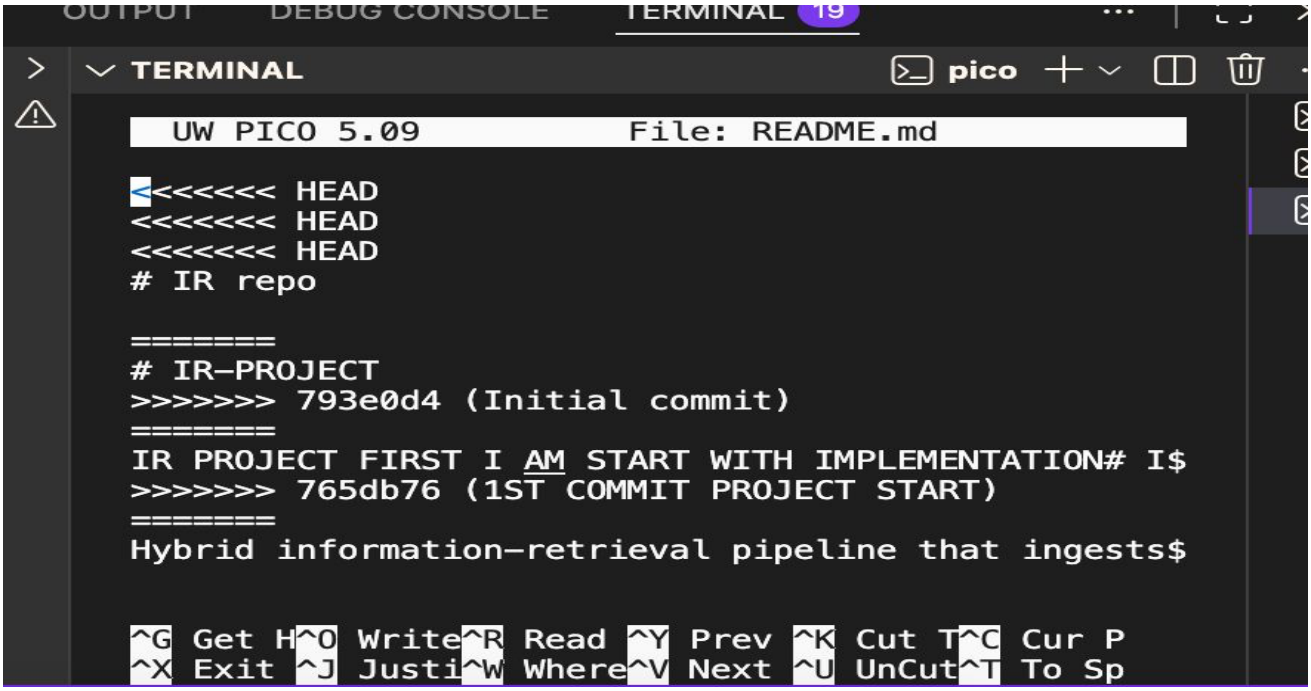




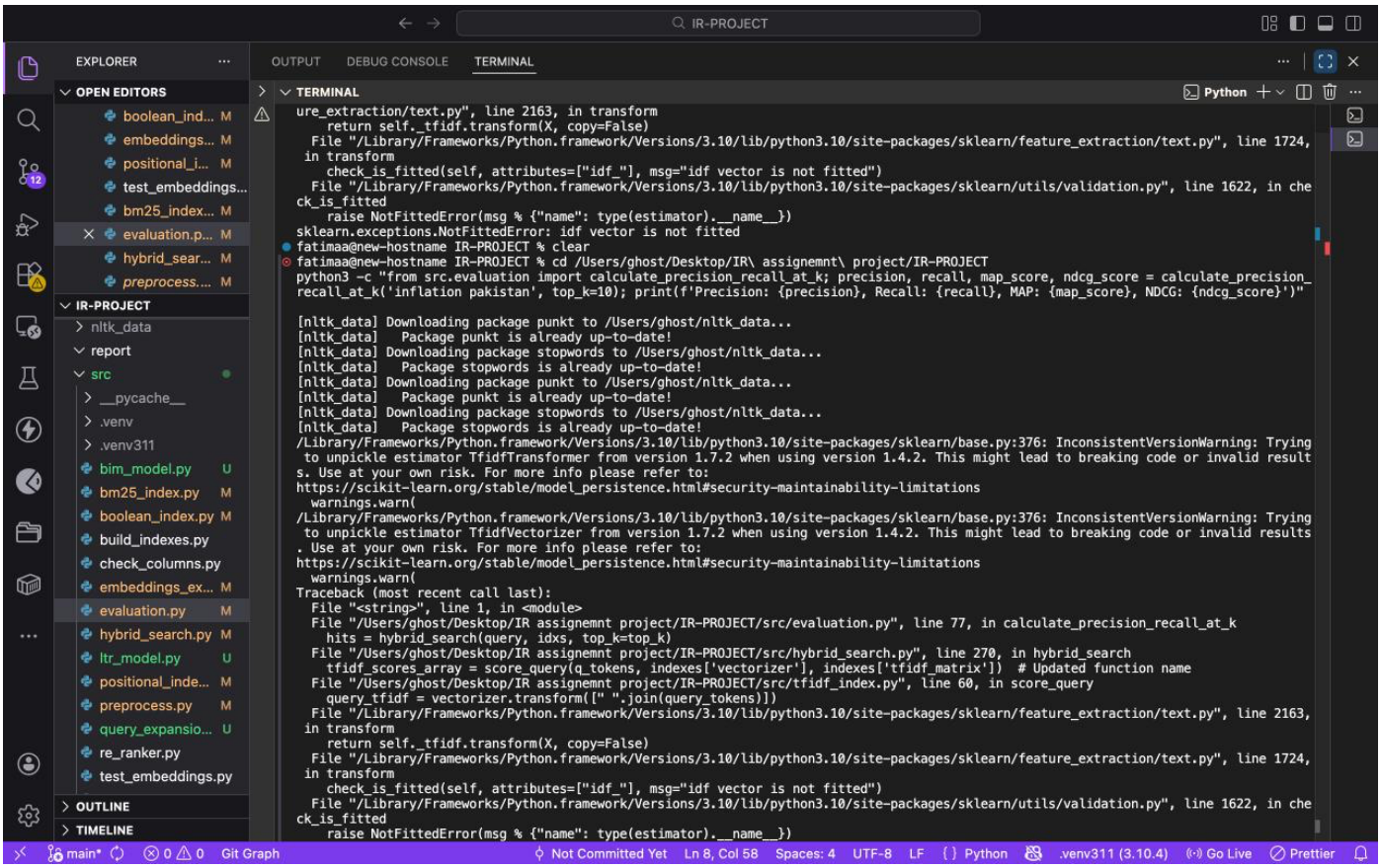
here is my error give guidelines how i resolve that error

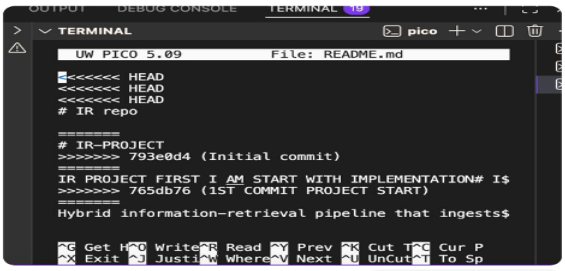
It appears that you're encountering the error `ModuleNotFoundError: No module named 'src'` when attempting to run the `build_indexes.py` module from the `src` folder.

This error typically occurs because Python cannot locate the `src` folder, possibly due



The conflict occurred because the repository already had an existing README.md file, which led to a merge conflict when trying to update it. I resolved this conflict by using ChatGPT to help guide me through the process.





how i resolve

The error you're seeing indicates that there are unresolved merge conflicts in your `README.md` file. The lines with `<<<<<<< HEAD`, `=======`, and `>>>>>>>` are conflict markers that Git uses to highlight where there are conflicting changes between two branches. Here's how to resolve it:

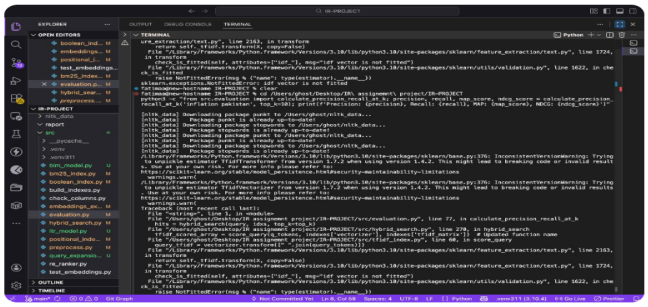
Steps to resolve the merge conflict:

1. **Open the file with conflicts**: In this case, it's `README.md`.

Python

```python
# (around line 146) until the tfidf_scores_array call.

def hybrid_search(query, indexes, top_k=10):
    q_tokens = preprocess_text(query)
    all_doc_ids = set(range(len(indexes['docs'])))

    # Step 1: Candidate Generation (Cleaned and Filtered)
    candidates = retrieve_and(q_tokens, indexes['boolean'])
    if not candidates:
        candidates = all_doc_ids
    candidates = list(candidates)

    # --- CRITICAL FIX: Ensures indices are valid (0 to N-1) ---
    max_doc_id = len(indexes['docs']) - 1
    candidates = [i for i in candidates if i <= max_doc_id]
    # ----------------------------------------------------------

    tfidf_scores_array = score_query(q_tokens, indexes['vectorizer'], indexes['tfi
    tfidf_scores = {i: float(tfidf_scores_array[i]) for i in candidates}

    bm25_scores_arr = score_query_bm25(q_tokens, indexes['bm25'])
    bm25_scores = {i: float(bm25_scores_arr[i]) for i in candidates}
# ... rest of the function continues here
```

Hybrid search error does not remove then I update the code. By the help of AI in this line the issue is I implemented that line



give possible solution to resolve that eror i know the error in which file but u guide me with possisble solutions of it

The error you're encountering seems to be caused by an issue related to a "NotFittedError" in your machine learning pipeline, particularly when using `TfidfVectorizer` in your code.

Here are a few possible solutions to resolve the error:

1. Ensure that the TF-IDF Vectorizer is properly fitted

7. Repo link

https://github.com/CodeFramework-Tech/IR-repo