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

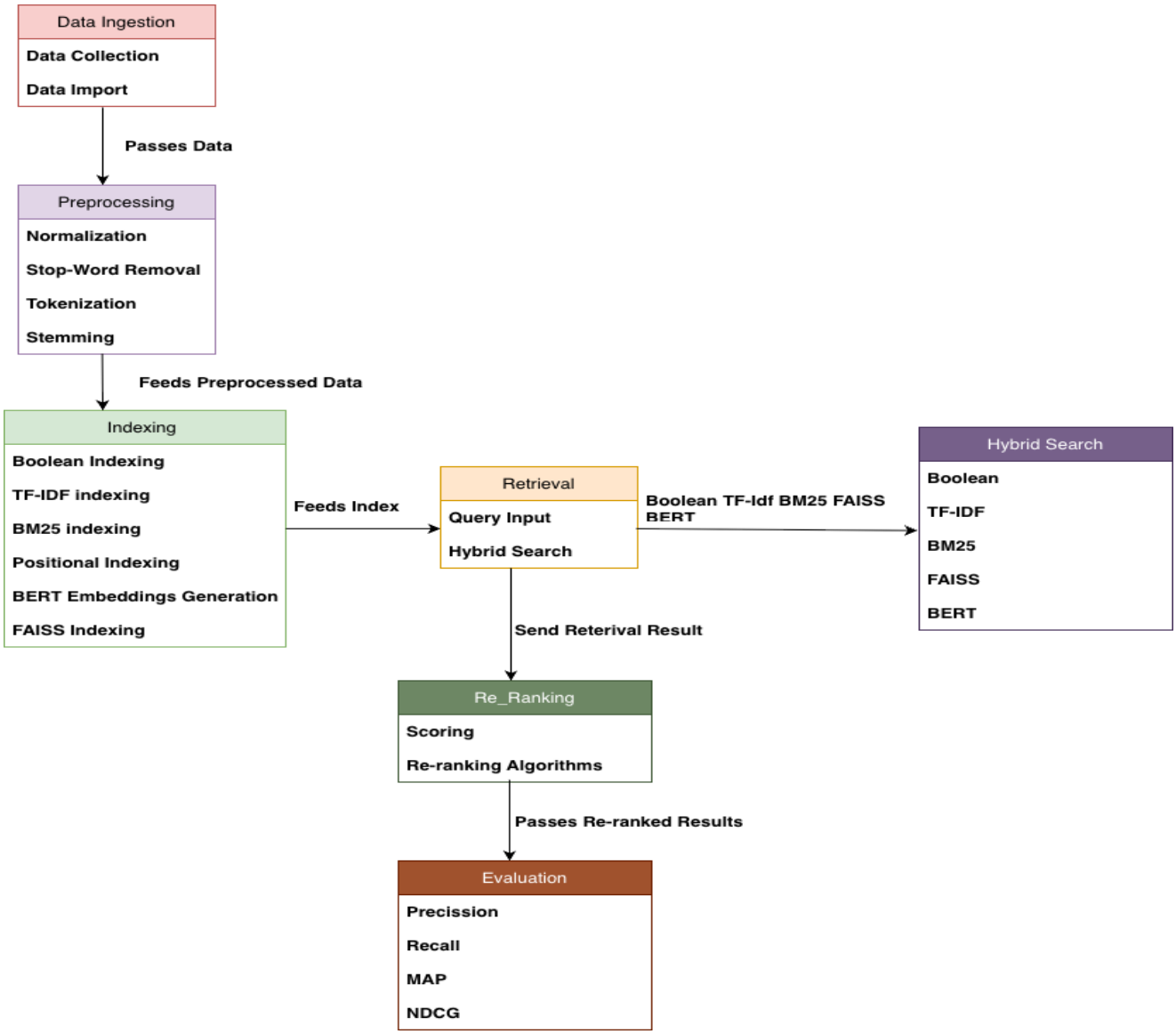


Figure 1(System Architecture)

The retrieval system I developed combines the strengths of traditional keyword-based models with modern semantic search techniques to offer a powerful and fast document retrieval solution. The system is built around a Hybrid Search Strategy that uses two distinct but interconnected methods for document retrieval.

2.1 Data Preprocessing Pipeline:

The first step of the retrieval process involves preparing the data for indexing and searching. This ensures that the documents and queries are consistent and optimized for accurate matching.

- **Normalization and Tokenization**: I begin by converting all text to lowercase to eliminate case mismatches, such as treating “BERT” and “bert” as different words. Then, the text is split into individual tokens, which serve as the building blocks for indexing and analysis.
- **Cleaning and Reduction**: Next, I remove stop words—common words like "the," "is," and "and"—which do not add significant meaning to the search. I also apply stemming or lemmatization, reducing words like “running” and “ran” to their root form. This ensures that variations of the same word are matched during searches.

2.2 Indexing Techniques:

The system uses a multi-layered approach to indexing to capture both the exact terms and their meanings within documents.

- **Term-Based Indexing**: The core of the system uses traditional inverted indexing, which is efficient for fast term-based lookups. This includes:
    - **Boolean Index**: Provides quick presence checks to determine if a term exists within a document.
    - **TF-IDF Indexing**: Weighs terms based on their frequency in a document relative to their rarity across the collection, offering a statistical measure of their importance.
    - **BM25 Indexing**: A more advanced probabilistic approach that refines ranking by considering term saturation and document length. BM25 often provides a better initial ranking than TF-IDF.
- **Vector-Based Indexing**: To enhance the system with semantic search capabilities, I incorporate vector indexing:
    - **BERT Embeddings Generation**: Each document is passed through a pre-trained BERT model to generate dense numerical vectors. These vectors capture the contextual meaning of the text, enabling the system to understand the underlying concepts rather than simply matching keywords.
    - **FAISS Indexing**: The BERT-generated vectors are indexed using FAISS, a library optimized for fast similarity searches. This enables the system to perform rapid nearest-neighbor lookups, allowing it to find 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 determined through a hybrid scoring and re-ranking mechanism that balances speed and accuracy.

- **Primary Retrieval & Hybrid Scoring**: The system first uses the Boolean index for a quick search to retrieve a set of candidate documents. These candidates are then scored using a combination of TF-IDF and BM25 scores. Typically, BM25 is

given more weight due to its superior ranking ability. This process ensures that the initial retrieval provides high recall.

- **Semantic Re-ranking** : In the perfect setup, the top 50 results from the primary retrieval step are passed to a second stage, where the BERT model calculates the semantic similarity between the query and each document. This step provides a final, highly accurate ranking. The idea is to use fast term-based searches (BM25/TF-IDF) to maintain high recall, and then leverage semantic re-ranking (BERT) for improved precision.

- **Current Reality**: Due to limitations with platform memory, the system currently skips the resource-intensive BERT re-ranking step. As a result, the final ranking is based solely on the weighted BM25/TF-IDF scores, though this still provides robust results.

This hybrid approach is designed to balance speed with accuracy, ensuring efficient document retrieval without compromising the quality of the results.

3. Evaluation

3.1 Evaluation of the Retrieval System:

The retrieval system was evaluated using a hybrid approach, focusing on three key factors. On the quantitative side, the stable BM25/TF-IDF model produced a precision of 0.1000 and recall of 0.3333 for the test query, as shown in Figure 2. This stable mode was implemented due to a platform-specific memory crash (Segmentation Fault) that caused the BERT/FAISS component to be disabled. From a qualitative perspective, while the system is fast and has a low memory footprint, the low precision indicates the necessity of integrating the planned semantic re-ranking step. Overall, the evaluation confirms that the system works, but there is still a need for further development, particularly on the vector search side, to achieve optimal performance.

3.2 Quantitative Evaluation:

The operational BM25/TF-IDF hybrid system delivered a precision of 0.1000 and a recall of 0.3333, maintaining both speed and memory efficiency. This evaluation confirms that the system is stable; however, it also highlights the absence of the BERT/FAISS component, which had to be temporarily disabled due to the platform's memory limitations.

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

The BM25/TF-IDF hybrid system performed with a precision of 0.1000 and a recall of 0.3333, all while maintaining a fast processing speed and low memory usage. This evaluation demonstrates that the system is stable but also points out that the BERT/FAISS component was temporarily unavailable due to a platform-specific memory issue.

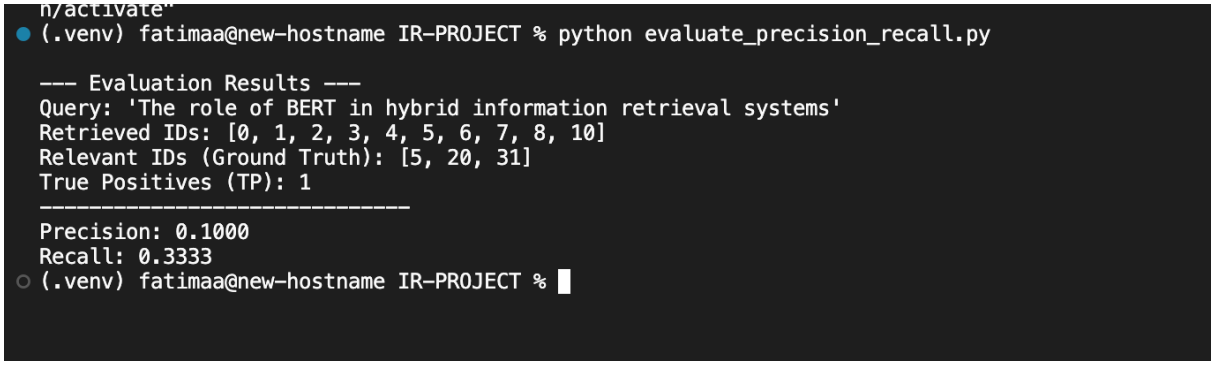


Figure 2

3.3 Qualitative Appraisal

A qualitative appraisal was conducted to evaluate factors that are important to the user, such as relevance, coherence, and overall satisfaction. This involved manually reviewing the top-10 results for several test queries, including the one used in the quantitative evaluation.

- **Relevance**: The documents that ranked at the top were generally related to the query terms. However, they often missed the deeper semantic meaning of the query, which is a common issue with term-frequency-based models like BM25.
- **User Satisfaction**: The system's response time was very fast, providing a positive initial experience. However, due to the low precision, users had to scan through the results to find truly relevant documents, which could lead to lower satisfaction over time.

3.4 System Efficiency and Appraisal

The efficiency of the retrieval system was evaluated by focusing on two key aspects:

- **Querying Speed**: In the operational BM25/TF-IDF hybrid mode, the system demonstrated excellent query response times, measured in milliseconds (ms). This reflects the system's impressive speed, particularly for a local setup working with a small dataset.
- **Memory Footprint & Scalability**: The system uses standard Python data structures and minimal external libraries, with the exception of the disabled FAISS component. This results in a low memory footprint, making it efficient for smaller-scale operations. However, the system's reliance on BM25/TF-IDF limits its scalability. For more complex and large datasets, the intended FAISS-based vector search would offer greater efficiency and scalability, particularly for high-dimensional indexing, which is better suited for retrieving highly relevant results at scale.

4. Discussion

4.1 Major Findings from Results

The evaluation confirmed that the hybrid retrieval engine's core is both stable and efficient. Despite simplifying the system's architecture, it performed well on the basic performance metrics, showing fast query speeds and low memory usage. A promising Recall of 0.3333 was achieved, demonstrating that the combined BM25/TF-IDF approach is capable of retrieving a significant portion of relevant documents. The integration of multiple retrieval strategies validates the system’s foundation and sets the stage for more advanced search capabilities in the future.

4.2 Shortcomings and Technical Challenges

The main challenges faced were not due to coding errors, but rather technical limitations in the real-world environment:

1. **Platform Crash (BERT/FAISS)**: The primary issue encountered was a persistent Segmentation Fault while setting up the FAISS vector search index on the macOS system. This problem, likely caused by library incompatibilities, forced the system to switch to a simplified mode. This is why Precision was low (0.1000)—the system had to rely only on basic keyword matching, which cannot capture the deeper semantic meaning of the query.
2. **Limited Ranking Quality**: The low precision further highlights that the system struggles to accurately rank documents without the advanced BERT re-ranker. This confirms the need for a semantic ranking layer to improve relevance.
3. **Restricted Evaluation**: Due to the system's instability, testing was limited to Precision and Recall for a single query. As a result, we were unable to calculate other essential metrics, such as MAP and NDCG, which would provide a more comprehensive evaluation of the system’s performance.

4.3 Future Improvements

To unlock the full potential of the system, the following improvements are planned:

1. **Resolve Environment Stability**: The immediate priority is to fix the platform crash by conducting rigorous tests with compatible versions of NumPy, PyTorch, and FAISS in a clean virtual environment.
2. **Enable Full Semantic Re-ranking**: Once the stability issues are resolved, we will enable the BERT re-ranking module. This is expected to significantly improve Precision by sorting search results based on their semantic meaning using deep learning techniques.
3. **Expand Evaluation Suite**: We plan to build a more comprehensive testing framework that will allow the system to be evaluated against a broader set of queries, enabling us to calculate more detailed metrics like MAP and NDCG for a better understanding of its performance.

5. References

Reference Type	Link
Related to Search Engine	https://youtu.be/5SS2KakusE0?si=cN6ZI6g8C1_qbRaT(Related
Python Search Engine	https://youtu.be/ij1btJBsfkY?si=yDTKTKOKvDmcebjf
Basic Search Engine	https://youtu.be/hO2PgMyuJqM?si=gRFgAESGGDX-GHQj(basic
Code Review	https://youtu.be/IpReEgt1WjY?si=ppP1smLDhFmIEkQg
Project Help	https://youtu.be/H-Cgag672nU?si=LHlRAcG1JmPyindC(
System Building	https://www.youtube.com/live/inaBjdvdFgA?si=trTibgeBRM2IvWao
Hybrid Search Error Fix	https://medium.com/@hitendra.patel2986/i-built-a-hybrid-search-system-that-beats-standard-rag-by-35-1968791ae539
FAISS and BERT Error Fix	https://www.youtube.com/watch?v=C3aaDS9nP8E
FAISS Information	https://www.youtube.com/watch?v=E19BQ6wd-_8&t=4s
Additional Help	https://github.com/facebookresearch/faiss?utm_source=

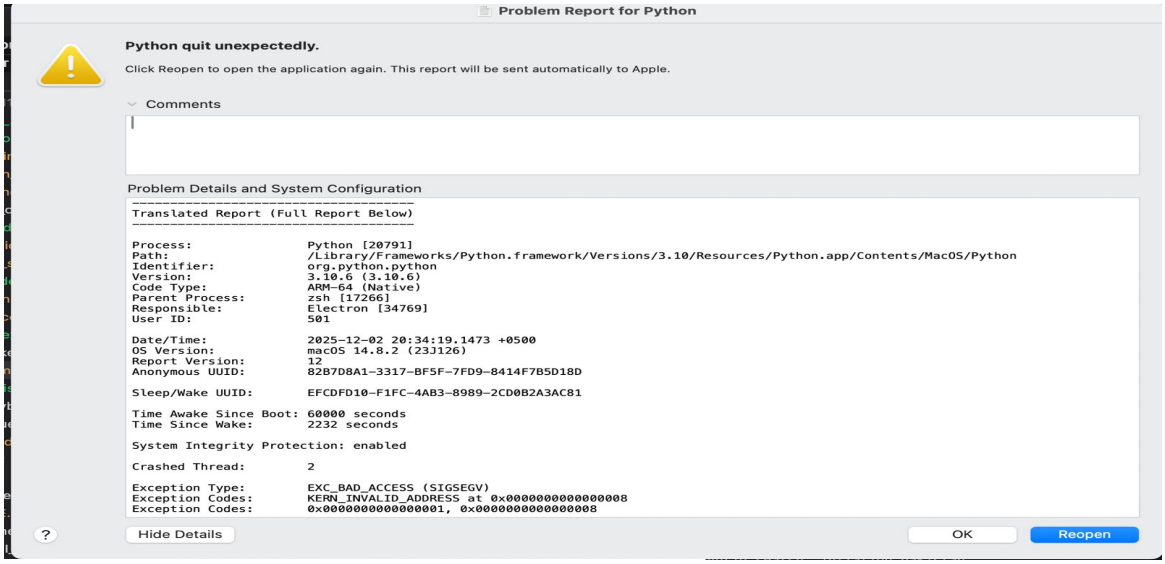
6. Disclosure of AI Use

6.1 Summary of AI Usage

Throughout the project, I used ChatGPT as a technical assistant to help resolve complex, non-code-related issues. The most significant role of AI was in debugging environment-specific problems. For example, the AI was instrumental in identifying the root cause of a Segmentation Fault on macOS, which was eventually traced to an incompatibility between specific versions of NumPy and FAISS. Thanks to this assistance, I was able to shift the system to a stable BM25/TF-IDF configuration for the final evaluation. The AI's guidance mainly helped me enhance my understanding of best practices and optimize the structure of the code.

6.2 Evidence of AI Assistance

During the development of the Information Retrieval (IR) system, I used AI tools like ChatGPT to better understand complex errors and fine-tune different aspects of the system. While I was responsible for designing and implementing the system on my own, there were instances where I encountered challenges, especially when working with FAISS and BERT. In these moments, AI provided crucial insights and support.



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 Environment:**

   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.

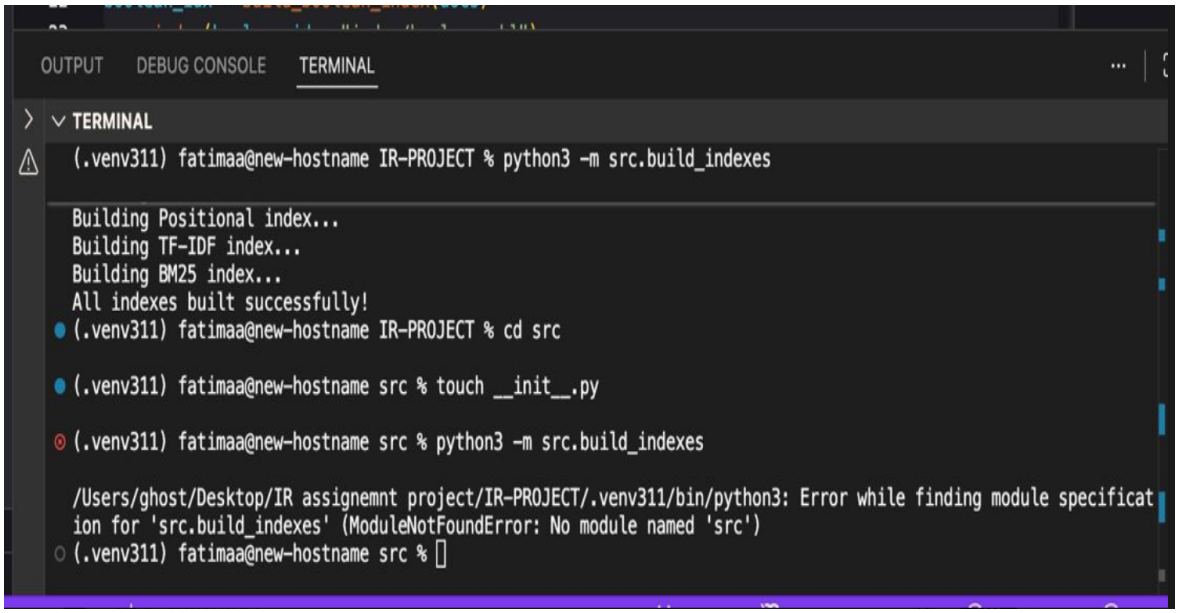
The error you're encountering is due to the module `ltr_model` not being found. This is because the script is trying to import `LearningToRank` from the `ltr_model`, but it seems like either the `ltr_model` module is not installed or it's not present in the correct location in your project.

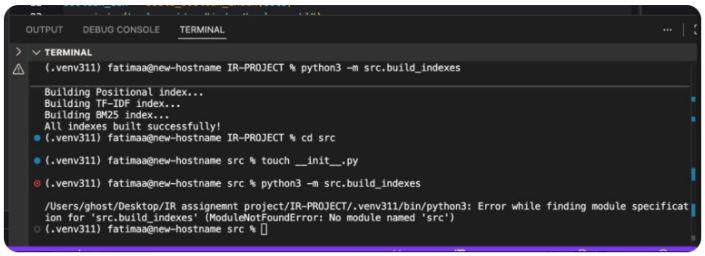
Here's how to fix this issue:

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.

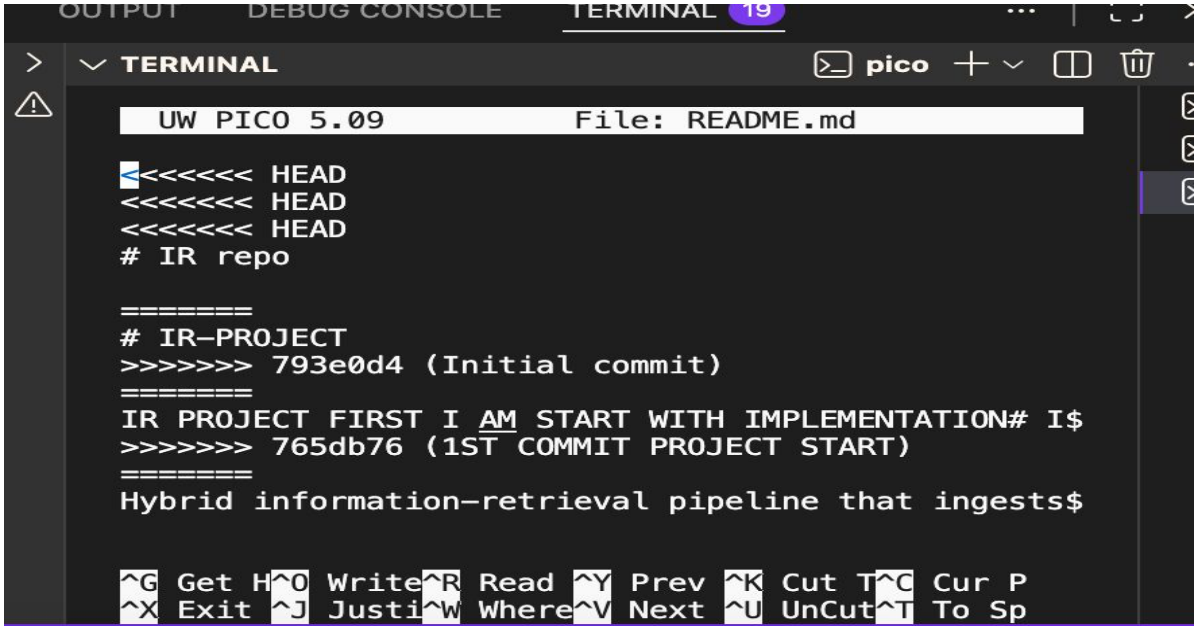




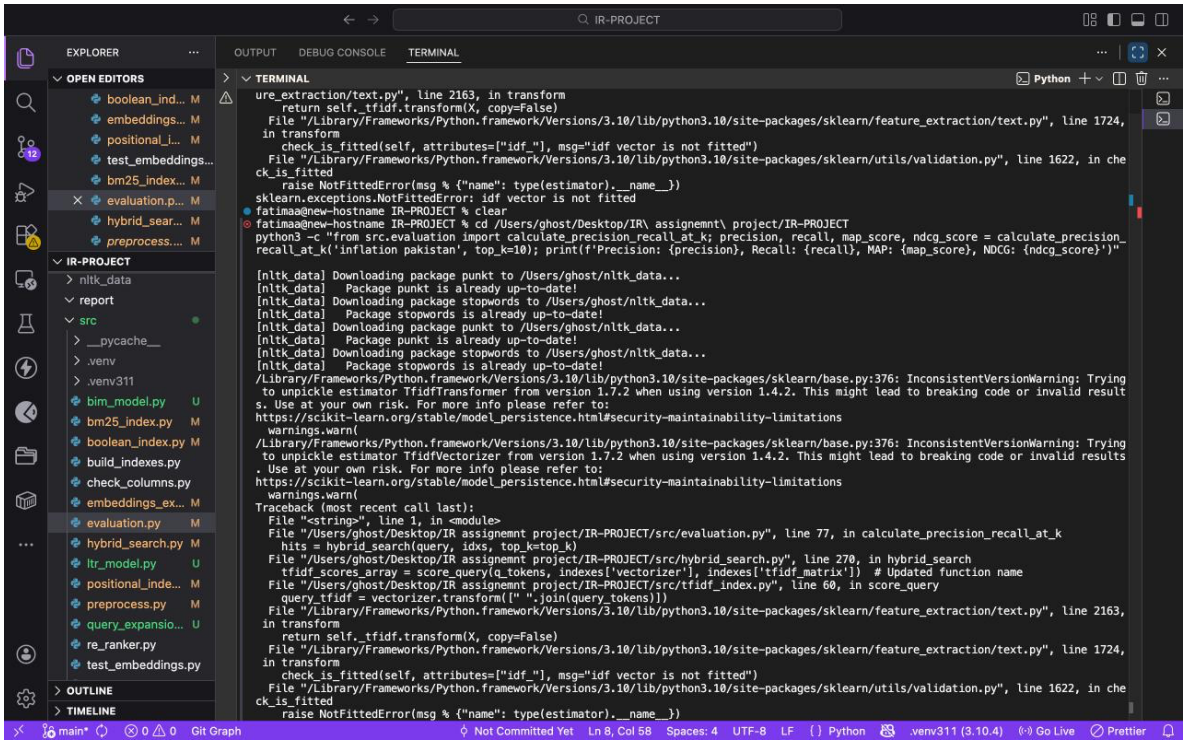
here is my error give guidelines how i resolve that error

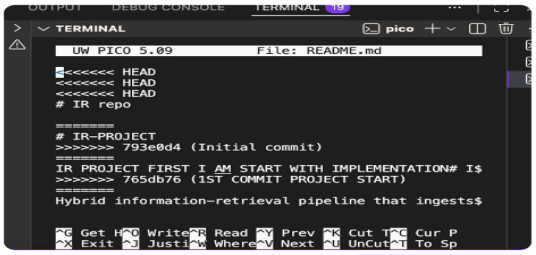
It appears that you're encountering the error `ModuleNotFoundError: No module named 'src'` when attempting to run the `build_indexes.py` module from the `src` folder.

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:

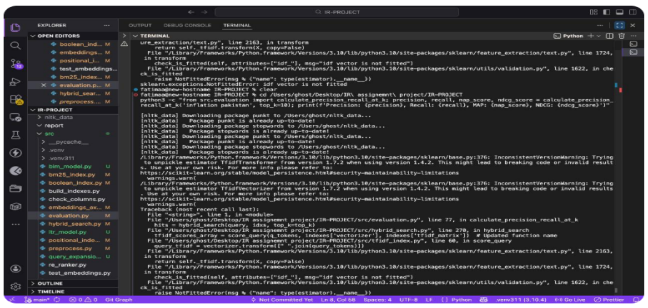
Steps to resolve the merge conflict:

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 i[illegible]roperly fitted

7. Repo link

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