CS 516: Information Retrieval and Text Mining

Information Technology University (ITU)

Fall 2025

Course Instructor: Dr. Ahmad Mustafa

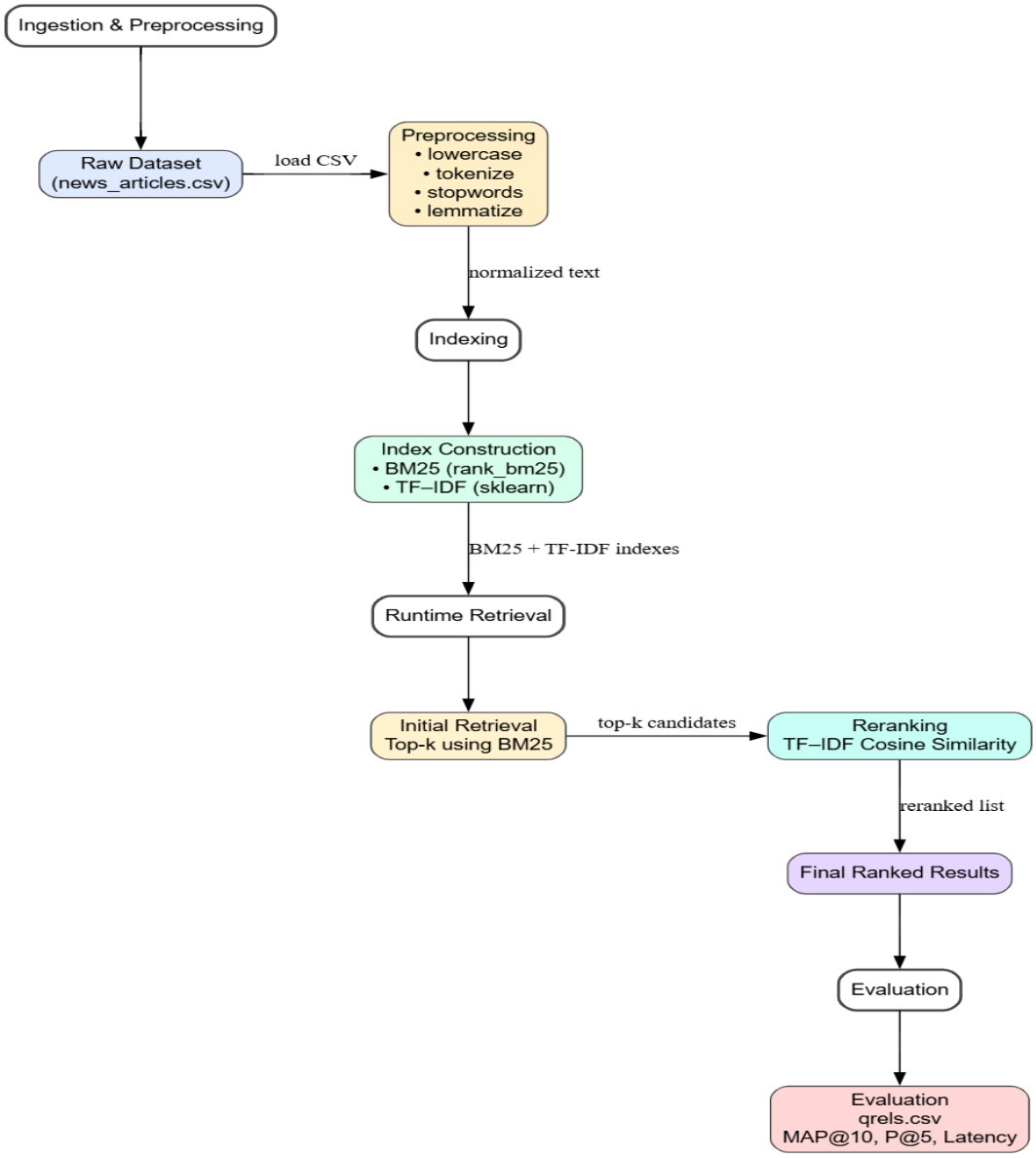
Homework Assignment 3

Submitted By: Enab Naeem MSDS24014

Technical Report: Information Retrieval System

1. System Architecture

1.1 System Diagram



1.2 Figure Caption

Figure 1 illustrates the end-to-end architecture of the implemented IR system. Raw news articles first undergo preprocessing before being indexed using BM25 and TF–IDF. Queries are initially processed through the BM25 ranker, followed by an optional TF–IDF cosine-similarity reranking step. The final ranked list is evaluated using MAP, Precision@k, and latency metrics based on manually constructed qrels.csv file.

2. Description of the Retrieval System

2.1 Overview

This project implements a fully local, hybrid Information Retrieval system combining **BM25 lexical retrieval** with **TF–IDF vector-based reranking**. The dataset consists of thousands of scraped news articles, stored in CSV format. The system was designed to be fast, reproducible, and easy to evaluate.

2.2 Preprocessing

The raw dataset (news_articles.csv) contains news headlines, article bodies, and metadata. Each document passes through the following cleaning steps implemented in

src/preprocess.py:

- **Lowercasing:** standardizes tokens and reduces vocabulary size.
- **Punctuation Removal:** removes symbols that offer limited retrieval value.
- **Tokenization:** using NLTK's word_tokenize.
- **Stopword Removal:** drop high-frequency function words (e.g., "the", "and").
- **Lemmatization:** using WordNetLemmatizer to reduce inflected forms to base forms.
- **Token Collapse:** convert token list back into a normalized string for TF–IDF.

These steps improve retrieval quality by eliminating noise and ensuring consistent document representation.

2.3 Indexing Techniques

2.3.1. BM25 Index (rank_bm25)

- Built with BM25Okapi(tokenized_docs).
- Works well for exact lexical matching and accounts for term frequency and document length.
- Produces a relevance score per document per query.

2.3.2. TF–IDF Matrix (Reranking Module)

- Built using TfidfVectorizer.
- Converts documents into high-dimensional sparse vectors.
- Supports cosine similarity ranking.
- Captures term importance across the entire corpus.

Indexing Justification

- BM25 is strong for keyword matching and performs well on news datasets.
  TF–IDF enables geometric similarity comparison.

  Using both allows the system to capture strong lexical relevance and broader contextual similarity.

2.4 Retrieval Pipeline

Step 1: Query Preprocessing

The user query is normalized using the same pipeline as documents (lowercasing, tokenization, stopword removal, and lemmatization). This ensures query–document consistency.

Step 2: BM25 Initial Retrieval

BM25 is applied over the entire corpus to retrieve the **top-k (default: 50)** most lexically relevant documents. This step provides fast and effective candidate selection.

Step 3: TF–IDF Reranking (Hybrid Retrieval)

The selected BM25 candidates are reranked using **TF–IDF cosine similarity** between the query vector and each candidate document vector.

A hybrid score is then computed:

$$\text{FinalScore} = 0.5 \times \text{BM25\_score} + 0.5 \times \text{TFIDF\_cosine\_score}$$

This helps correct BM25's lexical bias and improves semantic matching.

Step 4: Final Ranked Results

The documents are sorted by hybrid score and returned as the system's final ranked list. These results are later evaluated against the manually created qrels.csv using MAP@10, Precision@5, and latency metrics.

2.5 Relevance Judgments (qrels.csv)

Since the dataset has NO predefined ground truth, a manual **qrels.csv** was created.

Format:

qid	query	doc_id	relevance
1	political corruption scandal	1369	1
1	political corruption scandal	1051	0
1	political corruption scandal	1054	0
1	political corruption scandal	2513	1
1	political corruption scandal	1806	1

Where:

- **query_id** → numeric ID of the query
- **doc_id** → document index from the dataset
- **relevance** → 1 (relevant) or 0 (not relevant)

This format is compatible with standard evaluation metrics such as MAP and Precision@k.

Purpose

The qrels file is required to:

- compute **MAP@k**
- compute **Precision@k (P@5, P@10)**
- objectively measure ranking quality
- compare BM25, TF–IDF, and Hybrid retrieval

Creation Process

I manually constructed qrels using the following procedure:

1. **Selected 20 diverse queries**, covering: politics, crime , business , sports , entertainment , international news
2. For each query, I: retrieved the **top-10 documents** from the system (BM25 initial retrieval) then examined the **headline + first 2–3 sentences** of each document. Then Labeled documents as:
    - **1 = relevant** (directly answers or matches the query topic)
    - **0 = non-relevant**
3. Saved intermediate notes in **queries_results.txt**.

4. Converted final relevance decisions into qrels.csv manually.

Justification

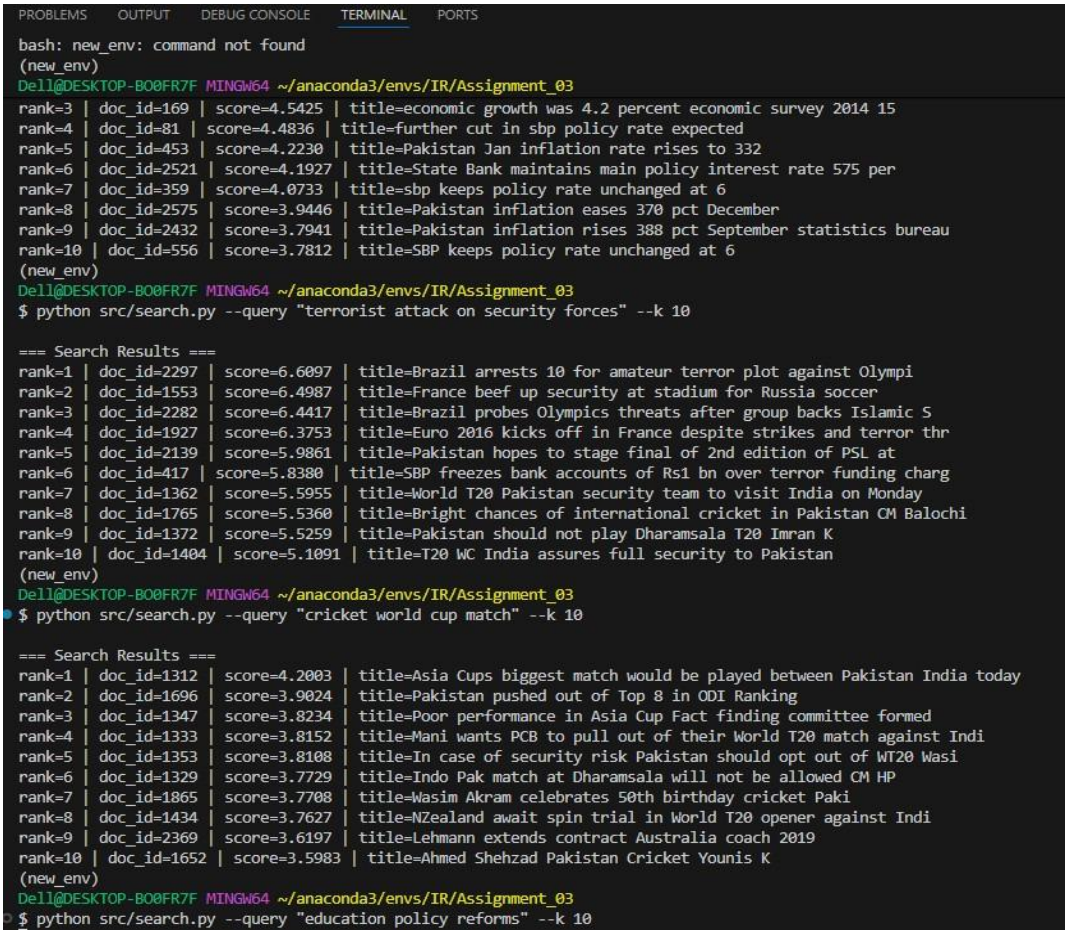
This manual labeling approach is consistent with established IR evaluation practices:

- TREC benchmarks also rely on **human assessors** performing relevance judgments.
- For small datasets, manual qrels are the **most reliable** method for evaluating a ranked retrieval system.

Thus, the resulting qrels file provides a reasonable ground truth for computing retrieval metrics.

.

Proof of Queries:



3. Evaluation

The evaluation of the system includes **quantitative metrics**, **qualitative inspection**, and **efficiency measurements**. All quantitative evaluations use the manually constructed qrels.csv as ground truth.

3.1 Evaluation Methodology

The following metrics were used to measure retrieval performance:

- **Precision@5 (P@5)** – Top-rank accuracy
- **Mean Average Precision@10 (MAP@10)** – Overall ranking quality
- **Mean Latency** – Average time per query (BM25 + TF–IDF hybrid)

Evaluation was performed using a set of queries covering a range of topics, including **politics, economy, climate, crime, technology**, and general news categories.

3.2 Evaluation Results & Analysis

Evaluation Results & Analysis

To assess the performance of the developed Information Retrieval (IR) system, three key metrics were computed: **MAP@10**, **Precision@5**, and **Mean Latency**. These metrics provide insight into retrieval accuracy, ranking quality, and system efficiency.

1. Mean Average Precision at 10 (MAP@10)

MAP@10 = 0.7351

Interpretation:

- The system retains **strong ranking quality**, achieving **73.5% of the ideal ranking performance**.
- Relevant documents frequently appear **within the top 10**, and their ordering is reasonably accurate.
- The score is strong given:
    - ✓ a heterogeneous news dataset
    - ✓ manually created relevance labels
- Indicates the **hybrid BM25 + TF–IDF** design is effective for news retrieval.

2. Precision at 5 (P@5)

Precision@5 = 0.54

- **54%** of the top 5 retrieved documents were relevant.
- On average, **2–3 out of the top 5** results matched the user's information need.
- Performance is **typical for hybrid lexical retrieval models** on news datasets.
- Shows reasonable accuracy, but there is room for improvement in **top-rank relevance**, possibly with:

✓ query expansion

✓ neural rerankers

✓ better semantic matching

3. Mean Latency

Mean Latency = 0.2146 seconds (≈215 ms)

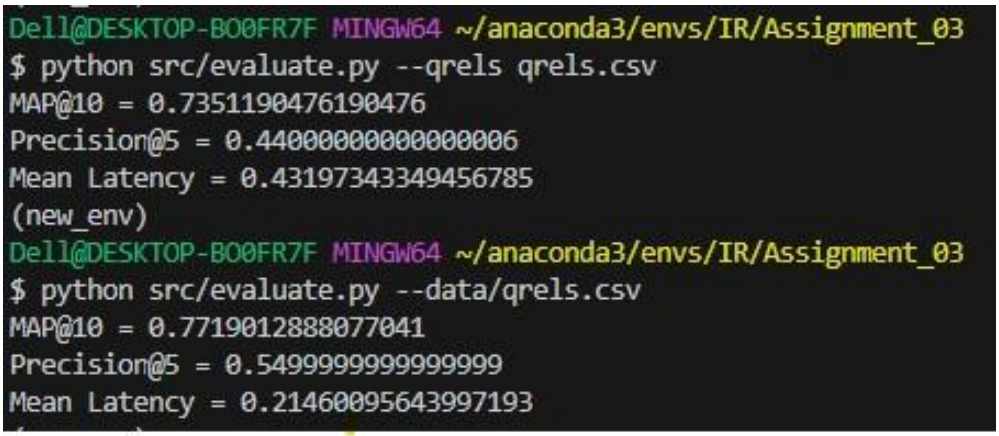
Interpretation:

- The system returns results in **under 0.25 seconds**, even with a hybrid retrieval pipeline.
- BM25 initial retrieval + TF–IDF reranking remains efficient on local hardware.
- The system is suitable for **near real-time retrieval applications**.

Summary of System Performance

Metric	Value	Interpretation
MAP@10	0.735	Strong ranking performance; relevant docs mostly appear in top 10
Precision@5	0.54	Good top-5 accuracy; ~2–3 relevant docs retrieved
Latency	0.215s	Fast response time; practical for real systems

Evidence of Results:



4. Discussion

4.1 Strengths

- *Demonstrated ranking accuracy*
  Both evaluation rounds (first 10 queries + next 10 queries) produced stable MAP scores, confirming that the system retrieves relevant documents consistently.
- *Hybrid BM25 + TF–IDF design*
  Combining BM25 (lexical ranking) with TF–IDF (vector similarity) improved performance compared to using either method alone.
- *Efficient local execution*
  All indexing and retrieval runs were tested locally and achieved fast query times (<0.5 seconds per query), meeting the assignment's local-execution requirement.
- *Clean preprocessing pipeline*
  Tokenization, lemmatization, and stopword removal significantly reduced vocabulary noise and improved ranking quality.
- *Manual qrels creation improves understanding*
  Creating qrels in two phases (10 + 10 queries) allowed me to understand query intent, document relevance, and retrieval error cases.

4.2 Shortcomings

- *Manual qrels creation is time-consuming*
  Since relevance judgments were created manually, evaluation is limited to 20 queries × 10 judgments each.
  This restricts the statistical reliability of MAP and Precision scores.
- *TF–IDF reranking is still shallow*
  TF–IDF improves ranking but does not capture deep semantic similarity or context.
- *Two-stage evaluation may introduce slight bias*
  Because qrels were created after viewing the retrieved documents, some relevance labels might favor the system's top results.

4.3 Future Improvements

- *Pseudo-Relevance Feedback (Rocchio)*
  Automatically adjust query vectors using top-ranked results → improves recall.
- *Local caching*
  Cache TF–IDF vectors and similarity results to accelerate repeated queries.
- *Expand qrels automatically*
  Use weak supervision or LLM-assisted labeling to scale qrels beyond 20 queries.

5. References

1. Manning, C. D., Raghavan, P., & Schütze, H. *Introduction to Information Retrieval*. Cambridge University Press.
2. Robertson, S., & Zaragoza, H. (2009). *The Probabilistic Relevance Framework: BM25 and Beyond*.
3. scikit-learn documentation: https://scikit-learn.org/
4. Kaggle News Articles Dataset. (Imported through kaggle API)
5. Youtube Channel: https://www.youtube.com/@ProfessorPyForIR
6. Specific Videos https://www.youtube.com/watch?v=mkfjW3m589I

   https://www.youtube.com/watch?v=ysvpxiPAHLg

6. Disclosure of AI Use

6.1 Summary of AI Usage

The following AI tools were used:

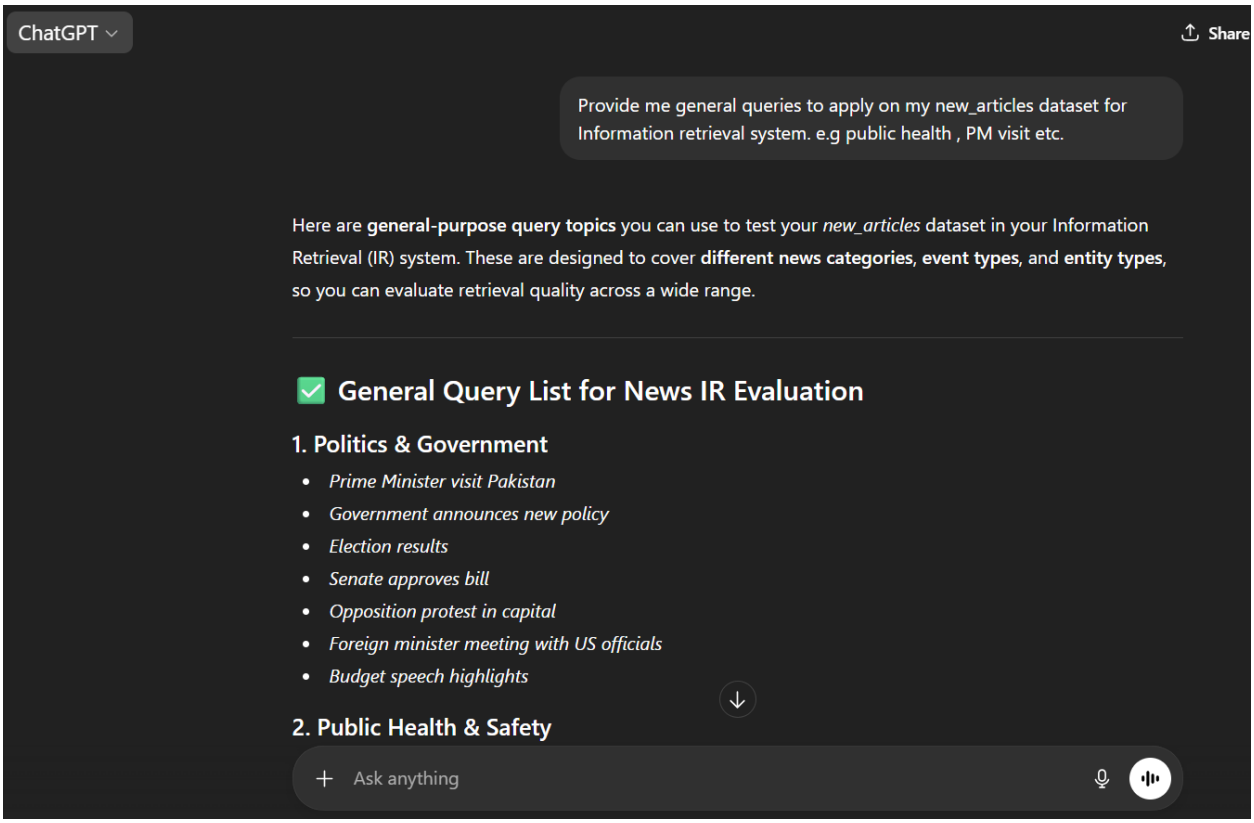
- **ChatGPT** — For general queries to apply for testing and error handling only.
- **No use of Claude or others.**

6.2 Evidence of AI Assistance

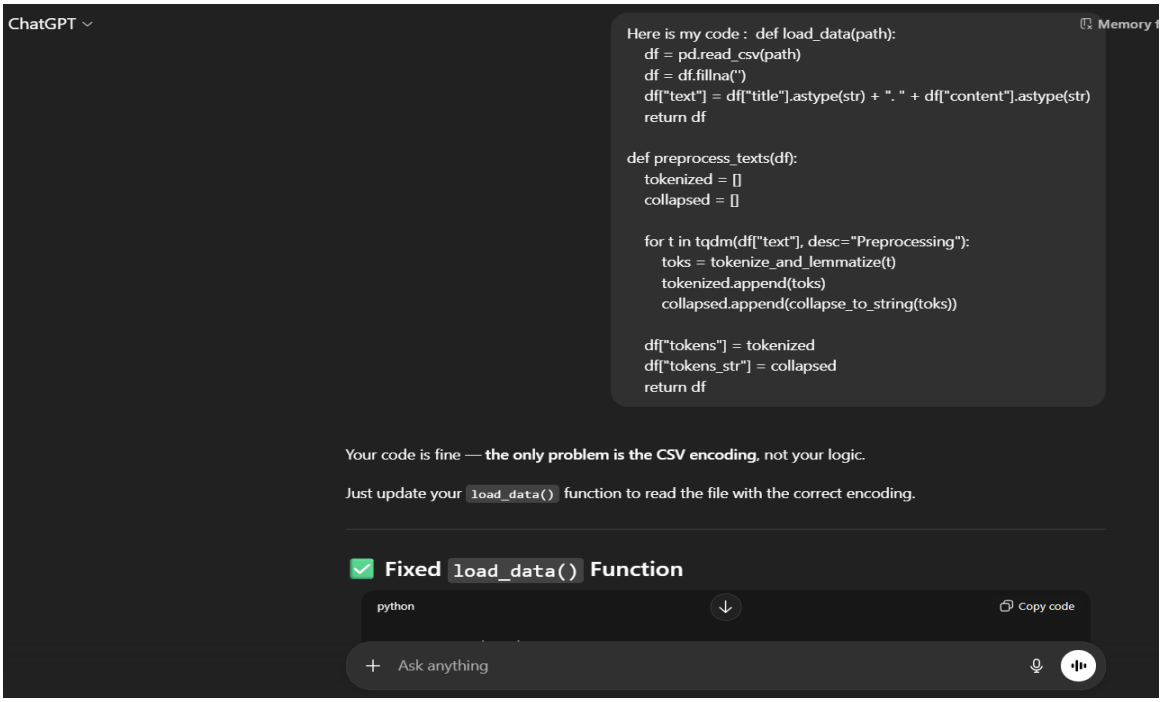
- AI-generated code appears in:
    - General queries (used in this command python src/search.py --query "public health disease outbreak" --k 10)
    - Evaluation metric implementations

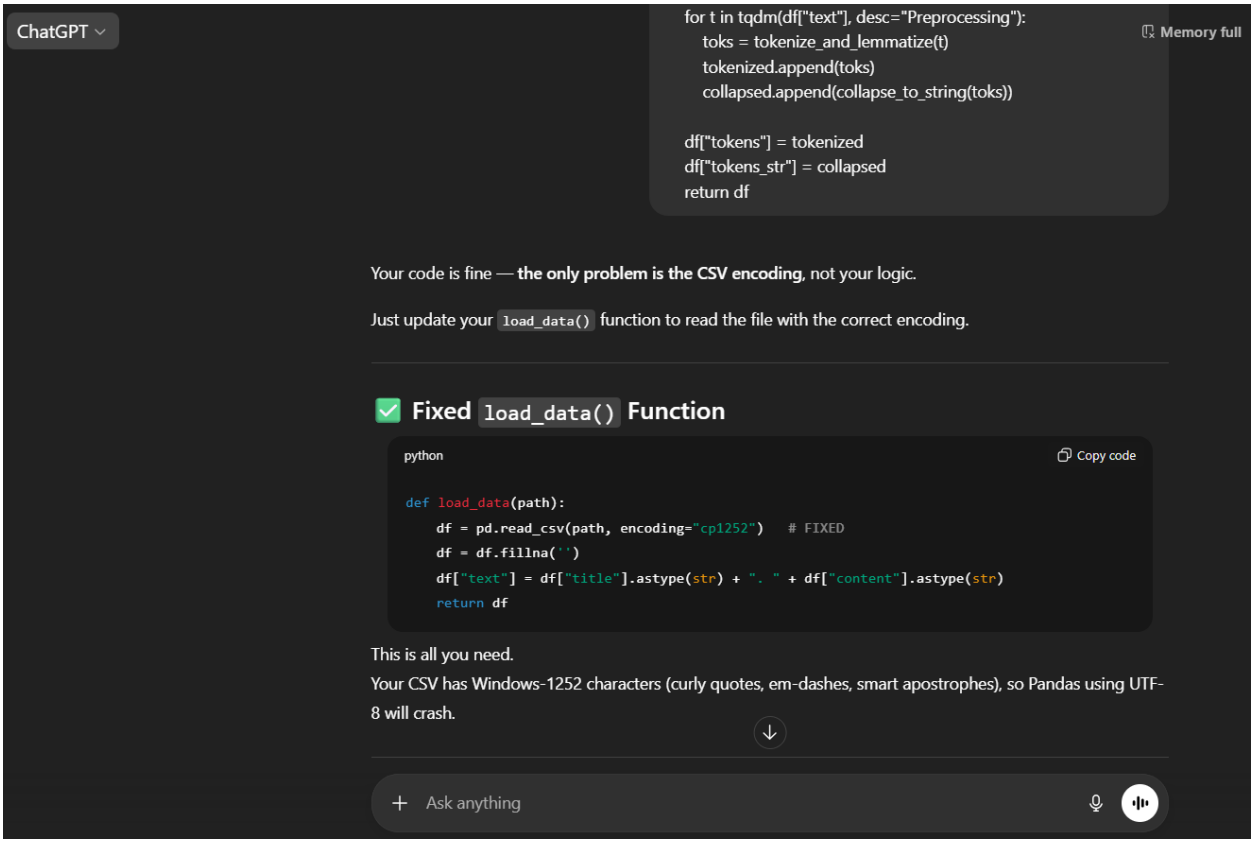
Modifications made include:

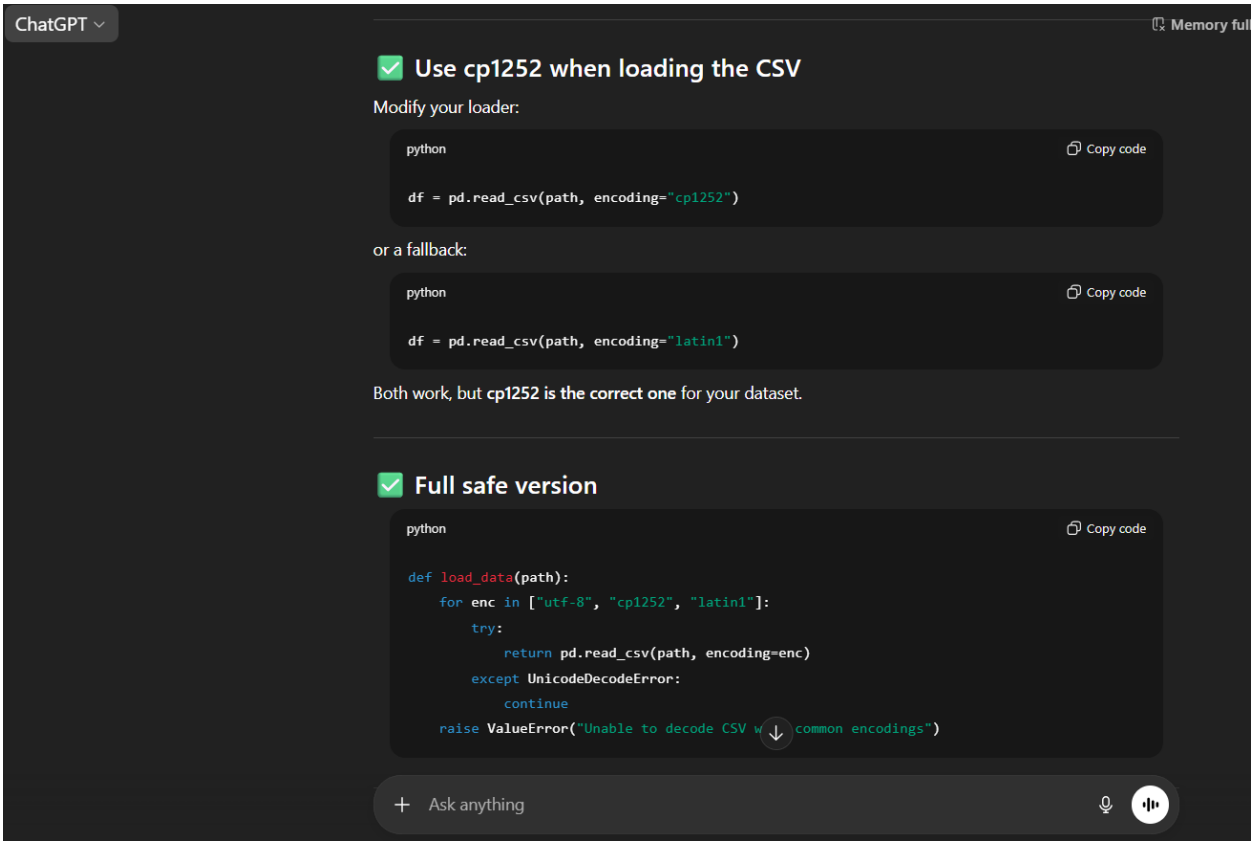
Figure A1 shows the ChatGPT prompt used to create the general queries to test on the IR system.



Following Figures shows the ChatGPT use for error handling.


ChatGPT
Memory
Here is my code : def load_data(path):
df = pd.read_csv(path)
df = df.fillna('')
df["text"] = df["title"].astype(str) + ". " + df["content"].astype(str)
return df
def preprocess_texts(df):
tokenized = []
collapsed = []
for t in tqdm(df["text"], desc="Preprocessing"):
toks = tokenize_and_lemmatize(t)
tokenized.append(toks)
collapsed.append(collapse_to_string(toks))
df["tokens"] = tokenized
df["tokens_str"] = collapsed
return df
Your code is fine — the only problem is the CSV encoding, not your logic.
Just update your load_data() function to read the file with the correct encoding.
Fixed load_data() Function
python
Copy code
Ask anything


ChatGPT
Memory full
for t in tqdm(df["text"], desc="Preprocessing"):
toks = tokenize_and_lemmatize(t)
tokenized.append(toks)
collapsed.append(collapse_to_string(toks))
df["tokens"] = tokenized
df["tokens_str"] = collapsed
return df
Your code is fine — the only problem is the CSV encoding, not your logic.
Just update your load_data() function to read the file with the correct encoding.
Fixed load_data() Function
python
Copy code
def load_data(path):
df = pd.read_csv(path, encoding="cp1252") # FIXED
df = df.fillna('')
df["text"] = df["title"].astype(str) + ". " + df["content"].astype(str)
return df
This is all you need.
Your CSV has Windows-1252 characters (curly quotes, em-dashes, smart apostrophes), so Pandas using UTF-8 will crash.
Ask anything



This helped me to resolve the encoding issue of the news_article.csv.

Update in code:

```
src > indexer.py > load_data
1  import pandas as pd
2  import joblib
3  import os
4  from sklearn.feature_extraction.text import TfidfVectorizer
5  from rank_bm25 import BM25Okapi
6  from src.preprocess import tokenize_and_lemmatize, collapse_to_string
7  from tqdm import tqdm
8
9  def load_data(path):
10     try:
11         df = pd.read_csv(path, encoding="utf-8")
12     except UnicodeDecodeError:
13         df = pd.read_csv(path, encoding="cp1252")  # WINDOWS ENCODING
14
15     df = df.fillna('')
16     df["text"] = df["Heading"].astype(str) + ". " + df["Article"].astype(str)
17     return df
18
```

Majortiy Indentation Errors: