# Information Retrieval Project

# Part B: Indexing with BERT and Query Interface Development

**Group 06**


Team Members:
G M Shahariar
Md Taukir Azam Chowdhury
Md. Olid Bhuiyan
Samiha Khan
Zabir Al Nazi


Date of Submission:
March 17th, 2025

# Table of Content

# System Overview

[Github Link]

## Architecture

The system consists of a dense indexing method:

- **BERT - Transformers (Dense Retrieval):**
  - Transforms text into **contextualized vector representations** for efficient **semantic matching**.
  - Tokenized and truncated Stack Overflow question titles and bodies to first 512 tokens.
  - Used the [CLS] token from BERT's last hidden state for embedding extraction.
- **FAISS (Efficient Similarity Search):**
  - Stores embeddings for **fast nearest neighbor retrieval**
  - **IndexFlatL2** - exact nearest neighbor with euclidean distance.
- **LLM-based Answer**: Using the exacted results, the system generates a natural answer.

## Processing Pipeline:

1. **Data Preprocessing**: Clean and tokenize data.
2. **Indexing**: Generate embeddings using BERT and store them in FAISS.
3. **Query Execution**: Retrieve relevant results from FAISS.
4. **Ranking & Output**: Rank and display results.
5. **LLM-based Answer**: Using the exacted results, the system generates a natural answer. (Additional)
6. **Web UI:** The output is displayed using a web UI. (Additional)

# Technologies Used

- Web Scraping: Scrapy (for collecting Stack Overflow data).
- Indexing and Search:
  - Sparse Retrieval: Pylucene (Inverted Index, Vector Space Model, Okapi BM25).
  - Dense Retrieval: BERT (🤗 Transformers) and FAISS for semantic search.
- Large Language Model: Qwen 2.5 3B Instruct (via Hugging Face API).
- Backend: Flask API for connecting search and AI models.
- Frontend: HTML, CSS, JavaScript for the user interface.
- Deployment: Hosted on a live server for real-time access.

## BERT Model & Indexing Schema

- **Model Choice:** BERT-base-uncased for better accuracy.
- **Passage Splitting:** Each passage is limited to 512 tokens.
- **Embedding Storage:** FAISS index for fast similarity search.

## Query Execution & Ranking

- **BERT Querying:** Converts the query into an embedding and retrieves the closest vectors using FAISS.
- **IndexFlatL2** - exact nearest neighbor with euclidean distance.

## Runtime Analysis

| Step | Time |
|---|---|
| BERT embedding generation (GPU) | 50 minutes |
| faiss index creation time for bert embeddings | 7.32 seconds |
| Query time | 0.101 seconds |

More Details:

```
cs242@class-046:~/cs242$ scp znazi002@bolt.cs.ucr.edu:~/bert_faiss_search.py ~/cs242/bert_faiss_search.py
znazi002@bolt.cs.ucr.edu's password:
bert_faiss_search.py
cs242@class-046:~/cs242$ python3 bert_faiss_search.py
🔹 loading bert tokenizer and model...
✅ bert model loaded in 0.4977 sec
🔹 loading embeddings and metadata...
✅ loaded 226340 embeddings with dimension 768.
⏱️embedding load time: 0.4108 sec
⏱️metadata load time: 6.5972 sec
🔹 loading existing faiss index...
⏱️faiss index load time: 0.7429 sec
⏱️total load time: 7.7512 sec


🔍 computing bert embedding for query: "When to catch java.lang.Error?"...

⏱️bert embedding computation time: 0.0774 sec

🔍 searching for similar questions...

⏱️faiss search time: 0.1015 sec
⏱️total search time: 0.1017 sec

🔹 result 1: When to catch java.lang.Error? (distance: 22.4866)
🔗 link: https://stackoverflow.com/questions/352780/when-to-catch-java-lang-error
🏷️tags: ['java', 'error-handling', 'exception', 'java', 'error-handling', 'exception']


🔹 result 2: When would I use uncaught_exception? (distance: 26.1349)
🔗 link: https://stackoverflow.com/questions/275249/when-would-i-use-uncaught-exception
🏷️tags: ['c++', 'exception', 'error-handling', 'c++', 'exception', 'error-handling']


🔹 result 3: Connect to BQuery from Python, best practice (distance: 30.4505)
🔗 link: https://stackoverflow.com/questions/79259176/connect-to-bquery-from-python-best-practice
🏷️tags: ['python', 'google-bigquery', 'google-cloud-colab-enterprise', 'python', 'google-bigquery', 'google-cloud-colab-enterprise']
```

### Findings:

- BERT is slower but retrieves semantically richer results.

### LLM-based Answers (Additional Contribution)

We employ the **Qwen 2.5 3B Instruct**, an instruction-following model. It is hosted on Hugging Face's API inference platform (https://huggingface.co/Qwen/Qwen2.5-3B-Instruct), allowing seamless integration with our retrieval pipeline. The model is leveraged to process and generate meaningful answers from the retrieved documents (comments containing solutions). Our intuition is that integrating a Large Language Model (LLM) in the error tag retrieval system will enhance the accuracy and relevance of responses provided to users based on their queries.

To generate relevant responses, the system follows a three-step workflow:

**Step 1**: Take the user query and documents retrieved by Bert as input

**Step 2**: The query and the retrieved documents are formatted into a structured prompt. We used the following prompt template -

*“””Using the information from the following documents, answer the question concisely:*

*Document 1: {text_of_retrieved_document_1}*
*Document 2: {text_of_retrieved_document_2}*
*Document 3: {text_of_retrieved_document_3}*
*Question: {user_query}*
*Answer:*
*“””*

**Step 3:** The system interacts with the Hugging Face API using the InferenceClient. The model call involves the following steps:

- The prompt is converted to a chat-style conversation template.
- A POST request containing the model name and chat template is sent to the API endpoint.
- The LLM generates a response, leveraging its pre-trained knowledge and the provided context. For generation, the system uses a max token of 1024 tokens.
- The response is streamed in chunks. The system iterates over the output chunks to collect the response gradually. Each chunk is extracted, cleaned (removing None values), and concatenated to form the full response. The final response is presented to the user in a coherent format.

# GUI Implementation (Additional Contribution)

## Features:

- User-friendly web interface
- Allows Lucene or BERT-based search

- Displays ranked solutions with AI-generated summaries
- Backend: Flask API connecting search & AI models

### Hugging Face API Usage:

- Used pre-trained LLM via API call
- Requires API key authentication
- Sends query & retrieved answers for processing

## Performance Evaluation with PyLucene

Although PyLucene is not used for indexing in this part, we evaluate the performance of BERT retrieval in comparison to traditional sparse retrieval methods like Lucene.

### Runtime Comparison

| Method | Indexing Time | Query Time |
|---|---|---|
| BERT + FAISS | 50 minutes | 0.101 seconds |
| PyLucene | 223.26 seconds | 0.22 seconds |

### Ranking Comparison

| Example Query | Top-1 BERT Result | Top-1 PyLucene Result |
|---|---|---|
| "MySQL group by error" | Logical error discussion | Syntax-related error post |

**Observation:**

- BERT captures intent better for ambiguous queries.
- PyLucene is more precise for well-formed queries with keywords.

### System Limitations

- **BERT Limitations:** High memory consumption, slower indexing.
- **PyLucene Limitations:** Does not understand synonyms and semantic similarity.
- **Deployment Challenge:** Hosting FAISS with large datasets requires GPU optimization.

# Deployment & Usage Instructions

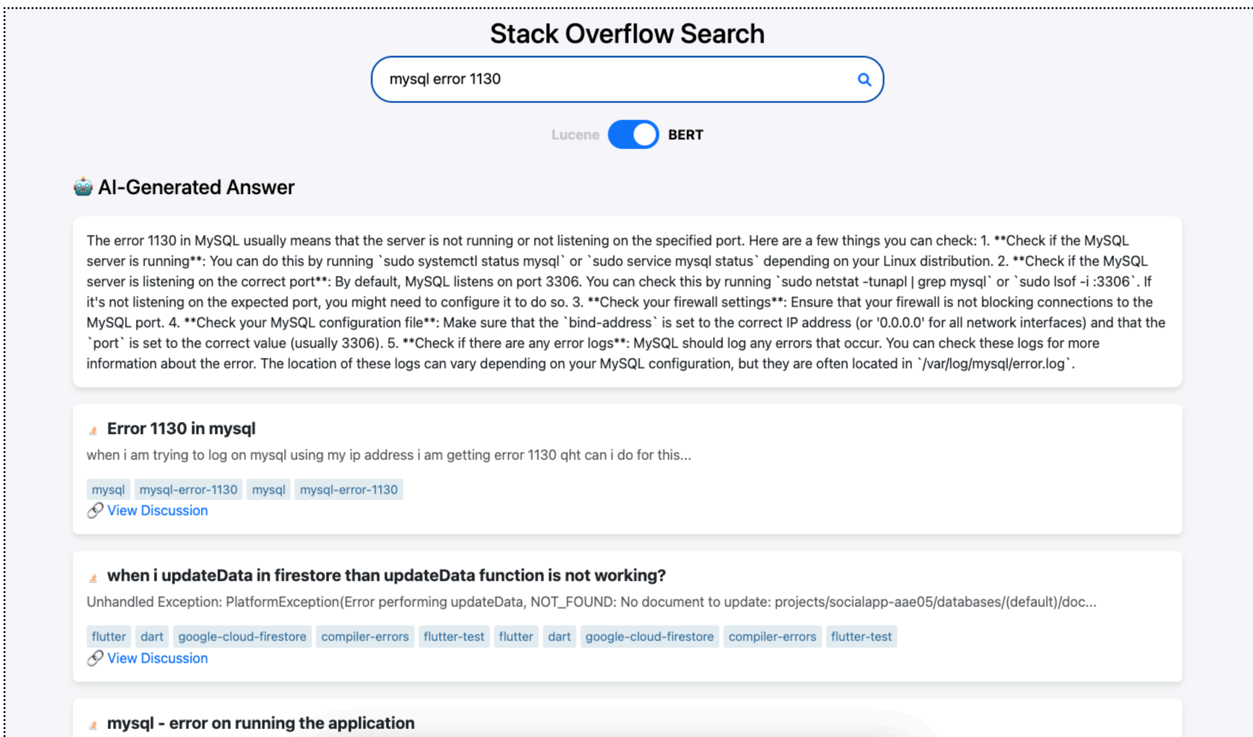
Run the code in terminal: main.py

### Web Interface (Flask-based)

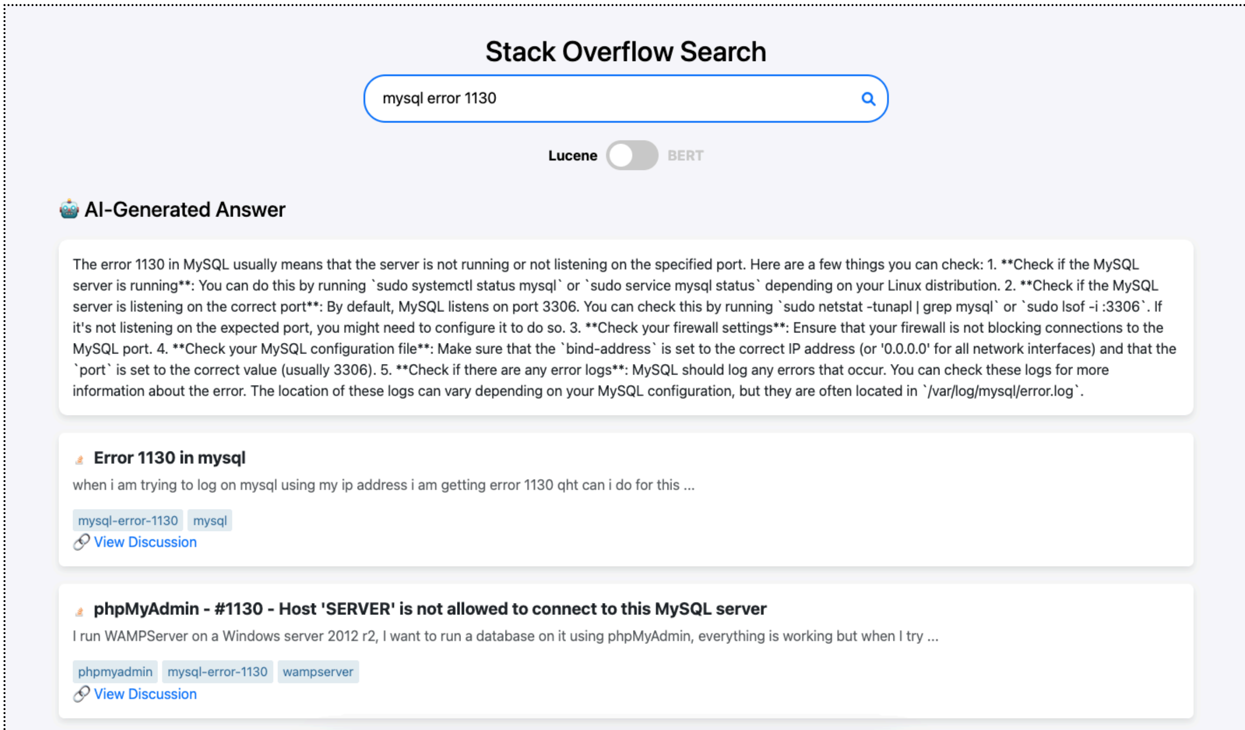
1. Start server: python3 app.py
2. Access at: http://169.235.31.51:8080/

# Screenshots & Visuals

Bert-Based Search:



Lucene-Based Search:




## Output displayed in terminal (With distance score):

```
🔍 computing bert embedding for query: "When to catch java.lang.Error?"...

⏱bert embedding computation time: 0.0774 sec

🔍 searching for similar questions...

⏱faiss search time: 0.1015 sec
⏱total search time: 0.1017 sec

🔹 result 1: When to catch java.lang.Error? (distance: 22.4866)
🔗 link: https://stackoverflow.com/questions/352780/when-to-catch-java-lang-error
🏷tags: ['java', 'error-handling', 'exception', 'java', 'error-handling', 'exception']


🔹 result 2: When would I use uncaught_exception? (distance: 26.1349)
🔗 link: https://stackoverflow.com/questions/275249/when-would-i-use-uncaught-exception
🏷tags: ['c++', 'exception', 'error-handling', 'c++', 'exception', 'error-handling']


🔹 result 3: Connect to BQuery from Python, best practice (distance: 30.4505)
🔗 link: https://stackoverflow.com/questions/79259176/connect-to-bquery-from-python-best-practice
🏷tags: ['python', 'google-bigquery', 'google-cloud-colab-enterprise', 'python', 'google-bigquery', 'google-cloud-colab-enterprise']
```

# Github Link:

We have uploaded our full code, with detailed description in the readme file. You can check the code here: GITHUB

# Contribution

**BERT Indexing & FAISS Implementation:**

- Research and selection of BERT model: Zabir Al Nazi, G M Shahariar
- Implementing BERT-based embedding generation: Md Taukir Azam Chowdhury, Md. Olid Bhuiyan, Samiha Khan
- Storing embeddings in FAISS and optimizing indexing: Zabir Al Nazi
- Improving indexing speed and memory optimization: G M Shahariar

**Query Interface Development:**

- Command-line interface implementation: Samiha Khan, G M Shahariar
- Web application development (Flask-based): Md Olid Bhuiyan, Md Taukir Azam Chowdhury
- Integration of BERT retrieval with FAISS: Zabir Al Nazi

**Performance Analysis & Ranking Comparison:**

- Collecting test cases and analyzing query outputs: Zabir Al Nazi
- Comparing runtime efficiency: G M Shahariar

**System Deployment & Testing:**

- Writing deployment scripts and instructions: Md. Olid Bhuiyan, Samiha Khan, G M Shahariar
- Testing system performance under different query loads: Md Taukir Azam Chowdhury, Zabir Al Nazi, Md. Olid Bhuiyan, Samiha Khan, G M Shahariar

**Reporting & Documentation:**

- Writing system architecture and indexing details: Md Taukir Azam Chowdhury, Samiha Khan, Zabir Al Nazi
- Explaining query execution and retrieval methods: G M Shahariar, Md Olid Bhuiyan
- Documenting obstacles, solutions, and system limitations: Samiha Khan

**LLM-Based Answers (Bonus):** G M Shahariar, Md Olid Bhuiyan