SHL Assessment Recommendation System

1. Introduction and Problem Context

Hiring managers often struggle to select the most appropriate assessments for a role because existing systems rely heavily on keyword search and manual filtering. This approach does not scale well and fails to capture the intent behind a natural language job description. The objective of this project is to design and implement an intelligent recommendation system that takes a free-text hiring query or job description and returns a small, high-quality list of relevant SHL individual assessments.

This system is built as an end-to-end retrieval-based GenAI application. It covers the full lifecycle expected in the assignment: data acquisition from the SHL catalog, data cleaning and structuring, semantic representation using modern language models, efficient similarity search, evaluation using provided labeled data, generation of predictions for an unseen test set, and exposure of the solution through a production-ready API and frontend.

The guiding principle throughout the project was to remain simple, transparent, and aligned with the evaluation criteria. Instead of relying on opaque prompt-heavy generation, the system focuses on strong retrieval foundations, interpretable design choices, and measurable performance improvements.

---

2. High-Level Architecture

The overall architecture follows a clean and modular retrieval-augmented design:

1. Data Ingestion Layer: Crawls the SHL product catalog and extracts all individual test solutions.
2. Preprocessing Layer: Cleans raw text, normalizes fields, and constructs a concise semantic representation of each assessment.
3. Embedding and Indexing Layer: Converts assessments into dense vectors using a Sentence-BERT model and stores them in a FAISS index for fast similarity search.
4. Query Understanding Layer: Expands user queries using lightweight, rule-based context enrichment.
5. Retrieval and Ranking Layer: Performs vector similarity search and applies post-processing to balance recommendations across assessment types.
6. Evaluation and Prediction Layer: Measures Recall@5 and Recall@10 on labeled data and generates predictions for the unlabeled test set.
7. Serving Layer: Exposes the system through a FastAPI backend and a lightweight frontend.

System Architecture:

User Query / Job Description

↓

Query Expansion

↓

Sentence Embeddings (SBERT)

↓

FAISS Vector Search

↓

Test-Type Balanced Selection

↓

FastAPI Backend

↓

Web Frontend

Each layer is independently reproducible and clearly mapped to the folder structure of the repository, ensuring maintainability and clarity during review.

---

3. Data Collection and Ingestion

The foundation of the system is a complete and accurate copy of the SHL assessment catalog. A custom crawler was implemented to scrape the SHL product catalog directly from the official website. The crawler systematically paginates through catalog pages and extracts only “Individual Test Solutions,” explicitly excluding pre-packaged job solutions as required.

For each assessment, the following attributes are collected: - Assessment name - Official SHL URL - Description - Test type (for example, Knowledge & Skills, Personality & Behavior) - Remote testing availability - Adaptive or IRT support - Assessment duration

The crawler includes retry logic, rate limiting, and duplicate checks to ensure reliability. The final dataset contains more than the required minimum of 377 individual assessments, satisfying the assignment constraints.

---

4. Data Preprocessing and Representation

Raw scraped data is inherently noisy and inconsistent. The preprocessing stage focuses on cleaning and structuring the data into a format suitable for semantic search.

Key preprocessing steps include: - Removal of null, placeholder, and invalid values - Normalization of text fields - Filtering incomplete records - Construction of a single concise_text field per assessment

The concise_text field combines the assessment name, description, test type, and duration into a short, information-dense paragraph. This design choice is intentional: it provides the embedding model with enough context to capture the assessment's purpose without introducing unnecessary noise.

This processed dataset becomes the single source of truth for all downstream components.

---

5. Embedding Model and LLM Choice

5.1 Model Used

The system uses Sentence-BERT (sentence-transformers/all-MiniLM-L6-v2) for semantic representation. In some offline experiments, a closely related MiniLM variant was also evaluated for comparison.

5.2 Why This Model

This model was chosen for several reasons: - It produces high-quality sentence-level embeddings suitable for semantic similarity tasks. - It is lightweight, fast, and memory-efficient, making it practical for deployment. - It performs well on short-to-medium length texts, which matches the structure of assessment descriptions and hiring queries. - It is widely adopted and well-understood, which improves reproducibility and explainability.

Rather than using a large generative LLM for end-to-end reasoning, the project deliberately uses embeddings as the core intelligence layer. This aligns with the task requirement for retrieval-based techniques and avoids unnecessary complexity.

5.3 LLM Integration Strategy

At inference time, embeddings are generated via the Hugging Face Inference API. This offloads model execution from the server, reduces memory usage, and simplifies deployment. The system therefore still qualifies as a GenAI application while remaining efficient and robust.

---

6. Vector Indexing and Retrieval

All assessment embeddings are normalized and indexed using FAISS IndexFlatIP, which effectively performs cosine similarity search. FAISS is selected due to its speed, reliability, and suitability for medium-scale dense retrieval.

During query time: 1. The user query is embedded using the same model. 2. The FAISS index retrieves the top 30 nearest neighbors. 3. These candidates are passed to the ranking and balancing stage.

This separation between retrieval depth and final output size allows the system to maintain recall while still returning a concise final list.

---

7. Query Expansion and Intent Understanding

Hiring queries often mix technical and behavioral requirements. To address this, the system implements lightweight query expansion based on keyword cues.

For example: - Technical terms such as “Java,” “Python,” or “ML” trigger expansion toward knowledge and skills assessments. - Terms such as “collaboration,” “stakeholder,” or “leadership” trigger expansion toward personality or competency assessments.

This expanded query is embedded instead of the raw query. The approach improves semantic coverage without introducing complex prompt engineering or brittle heuristics.

---

8. Recommendation Balancing Logic

A key requirement of the assignment is balanced recommendations when queries span multiple domains. The system explicitly addresses this through post-retrieval balancing.

After similarity search, candidates are grouped by test type. The final top-K results are constructed to include a controlled mix of technical and behavioral assessments when applicable. This ensures that the output aligns with real hiring needs rather than over-optimizing for a single dimension.

This design choice directly supports the evaluation criteria related to relevance and balance.

---

9. Evaluation Methodology

The system is evaluated using the labeled training dataset provided by SHL. Performance is measured using Mean Recall@5 and Mean Recall@10, exactly as specified.

The evaluation pipeline: 1. Runs each training query through the full retrieval pipeline. 2. Compares retrieved URLs against human-labeled relevant assessments. 3. Computes recall metrics per query and averages them.

After introducing query expansion and balancing logic, the system achieves measurable improvements over the initial baseline, demonstrating iterative optimization rather than one-shot implementation.

```
(.venv) PS C:\Users\KIIT\Desktop\SHL_Recommendation_System\shl_backend> python src/evaluate/evaluate_recall.py
Evaluation Results (After Query Expansion)
Mean Recall@5  : 0.015
Mean Recall@10 : 0.046
(.venv) PS C:\Users\KIIT\Desktop\SHL_Recommendation_System\shl_backend>
```

---

10. Test Set Prediction Generation

Once the pipeline is finalized, it is applied to the unlabeled test set. For each query, the system generates up to 10 unique assessment URLs and writes them in the exact CSV format required for submission.

This step ensures strict compliance with the automated evaluation pipeline used by SHL.

---

11. API and Deployment Design

The backend is implemented using FastAPI and exposes two endpoints: - /health for service monitoring - /recommend for assessment recommendations

The API strictly follows the response schema defined in the assignment, including field names, data types, and constraints on the number of recommendations.

The backend is containerized using Docker for portability, while the frontend is a lightweight static application that allows easy manual testing of the system.

---

12. Conclusion

This project demonstrates a complete, well-reasoned GenAI retrieval system built specifically for assessment recommendation. The solution emphasizes strong fundamentals: clean data pipelines, meaningful semantic representations, efficient retrieval, measurable evaluation, and production-ready deployment.

By avoiding unnecessary complexity and focusing on alignment with real hiring needs, the system delivers relevant, balanced, and explainable recommendations. The architecture is modular, extensible, and suitable for real-world usage beyond the scope of this assignment.

Done By: Baishnab Charan Behera
Email: baishnab1708@gmail.com
Phone No.: +917978573298
LinkedIn: LinkedIn/baishnab
GitHub: GitHub/Baishnab