# SHL ASSESSMENT

# RESEARCH AI INTERN


# SIDDHARTH SARKAR


Before building the rag pipeline we need to crawl the data for individual test solutions from shl's product catalog and save it into a csv file for further analysis

Sample code snippet:

```python
import time
import math
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin
```

```python
BASE_URL = "https://www.shl.com"
CATALOG_URL = f"{BASE_URL}/products/product-catalog/"
HEADERS = {
    "User-Agent": "Siddharth/1.0 (contact: siddharth11.sarkar@gmail.com)"
}
```

```python
def get_soup(url, params=None):
    resp = requests.get(url, params=params, headers=HEADERS, timeout=15)
    resp.raise_for_status()
    return BeautifulSoup(resp.text, "html.parser")
```

```python
def extract_last_page_for_individual_tests(soup):
    """
    On the first catalog page, find the 'Individual Test Solutions' section,
    then read the pagination numbers below it to get the last page.
    """
    # Find the text node / heading that contains "Individual Test Solutions"
    header_text = soup.find(string=lambda t: t and "Individual Test Solutions" in t)
    if not header_text:
        raise RuntimeError("Couldn't find 'Individual Test Solutions' header")

    # The pagination block is a little bit below; simplest robust approach:
    # look for all links whose text is a digit and take the max.
    page_nums = []
    for a in soup.find_all("a"):
        text = (a.get_text() or "").strip()
        if text.isdigit():
            page_nums.append(int(text))

    if not page_nums:
        raise RuntimeError("No pagination numbers found")

    return max(page_nums)
```

```python
def extract_tests_from_page(soup):
    """
    Extract all tests on a page where type=1 (Individual Test Solutions).
    On those pages, the only catalog list present is the individual tests list,
    so we can simply grab all product-catalog/view links.
    """
    tests = []

    for a in soup.find_all("a", href=True):
        href = a["href"]
        if "/products/product-catalog/view/" in href:
            name = a.get_text(strip=True)
            if not name:
                continue

            full_url = urljoin(BASE_URL, href)
            # Nearby sibling text often includes the category codes like "K P S"
            codes = a.next_sibling
            codes_text = ""
            if codes and isinstance(codes, str):
                codes_text = codes.strip()

            tests.append({
                "name": name,
```

## 1. Problem Understanding

The objective of the project was to develop an **AI-powered recommendation system** capable of retrieving the most relevant **assessment URLs** based on a user query. The solution had to:

- Understand requirements that span multiple skill domains (technical + behavioral + managerial)
- Retrieve and recommend diverse and relevant assessments
- Ensure **non-repetitive results**
- Support **evaluation and performance benchmarking** using **Mean Recall@K**
- Improve accuracy iteratively based on the **Train-Set dataset**

The challenge was ensuring that the system does not hallucinate or recommend repeated content but instead produces **meaningful, diverse recommendations** aligned with expected results

## 2. Solution Architecture

### 2.1 Retrieval-Augmented Generation (RAG) Pipeline

The final pipeline included:

| Component | Description |
|---|---|
| Document Loaders | Loaded catalog assessments with names + URLs |
| Embedding Model | SentenceTransformers all-mpnet-base-v2 |
| Vector Database | Pinecone (Cosine similarity metric) |
| Retriever | Max Marginal Relevance (MMR) for diversity |
| LLM | Hugging Face model integrated with LangChain |
| Ranking Logic | Return top-K distinct URLs (unique enforced) |

**Pipeline Flow**

1. Query input
2. Embedding → vector search in Pinecone
3. Retrieval of diverse relevant assessments using **MMR**
4. LLM reranking and structured formatting enforcement
5. Output: Top K recommendations with **unique URLs**

**3. Performance Evaluation Strategy**

**Metric Selected: Mean Recall@K**

Since the goal was to retrieve **all expected relevant assessments**, Recall@K was the correct primary metric.

$$Recall@K = \frac{\text{Relevant items in Top K}}{\text{Total Relevant Items}}$$

$$MeanRecall@K = \frac{1}{N}\sum_{i=1}^{N} \quad Recall@K_i$$

**Evaluation Setup**

- **TrainData123.xlsx** → stored predicted URLs (1 URL per row)
- **Gen_AI Dataset.xlsx** → stored ground-truth relevant URLs (1 row per URL)
- K = 10

- Normalized all URLs (lowercase, remove query strings, remove slashes) to avoid false mismatches

# 5. Optimization Efforts

| Problem Identified | Optimization Applied | Result Improvement |
|---|---|---|
| Duplicate documents in index | Stopped re-upserting after first load (`from_existing_index`) | Cleaner ranking & better recall |
| Returned repeated URLs | Enforced unique URLs through LLM prompt + post-processing | 5 distinct recommendations |
| Similarity search lacked diversity | Switched to **MMR retriever** (`search_type="mmr"`) | More coverage across domain |
| Evaluation false negatives due to formatting | Added URL normalization | Correctly counted matches |
| Top results missed relevant items | Increased `fetch_k` and tuned `lambda_mult` | More exploration depth |

**USED LLAMA-3.1-8b-instant** – Specifically this model was chosen for this RAG bases application due to its strong balance of performance and cost-efficiency, optimized size for consumer hardware, and robust community support and ecosystem maturity.

Embedding model used **sentence-transformers/all-mpnet-base-v2**

**effective for RAG**

- **Semantic understanding:** The model is specifically designed to convert sentences and paragraphs into a vector space where the semantic meaning of the text is captured, which is crucial for finding relevant information in a RAG system.
- **Large-scale training:** It was fine-tuned on a massive dataset of over 1 billion sentence pairs using a self-supervised contrastive learning objective, which enhances its ability to capture the nuances of sentence-level meaning.

- **High performance:** For English RAG applications, it offers a strong balance of performance, scalability, and usability.
- **Broad applicability:** It works well for core RAG tasks like semantic search and information retrieval, allowing it to find relevant information based on a query.