Technical Approach: SHL Assessment Recommendation System

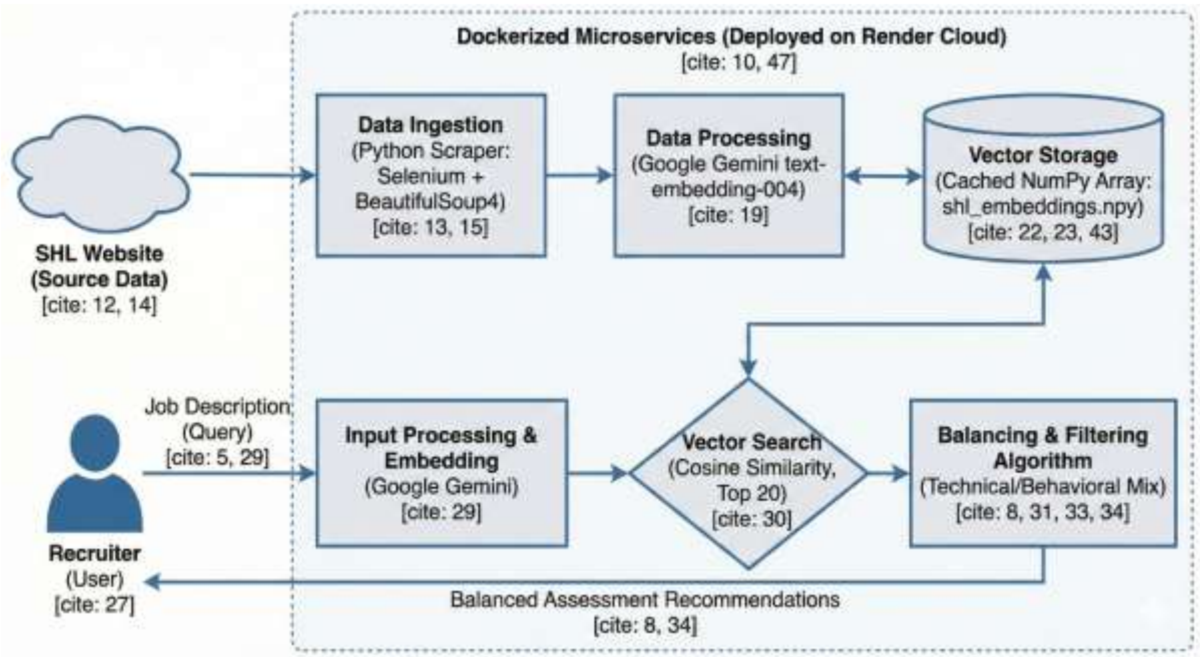
Submitted by: Vipul Patil | Date: December 18, 2025

1. Summary

The objective of this project was to engineer a robust, AI-driven recommendation engine capable of mapping natural language Job Descriptions (JDs) to relevant assessments from the SHL product catalog[1]. Traditional keyword-based search often fails to capture the nuance of hiring requirements—for example, mapping "team player" to a "Personality Test" rather than a technical exam[2].

To address this, I developed a Retrieval-Augmented Generation (RAG) pipeline using Google Gemini Embeddings and Vector Search[3]. The system scrapes real-time data from the SHL website, processes semantic intent, and applies a proprietary "balancing algorithm" to ensure recruiters receive a holistic mix of Technical (Knowledge) and Behavioral (Personality) assessment recommendations.

2. System Architecture



The solution is built on a modular Microservices architecture, containerized with Docker for seamless deployment[5].

A. Data Ingestion & Pipeline

- **Source:** The SHL Product Catalog (shl.com)[6].
- **Tooling:** A custom Python script utilizing Selenium and BeautifulSoup4[7].
- **Challenge:** The catalog uses dynamic JavaScript loading ("Load More" buttons) which prevents simple HTTP requests[8].
- **Solution:** The scraper automates a headless Chrome browser to scroll and click through pagination, extracting 377+ unique assessments[9].
- **Data Points Extracted:** Assessment Name, URL, Description, Duration, Test Type (categorized into "Knowledge & Skills" vs. "Personality & Behaviour"), and Remote Testing capability[10].

B. Vector Embeddings (The "Brain")

Once scraped, the textual descriptions of the assessments are converted into high-dimensional vectors11.

- **Model Used:** Google Gemini text-embedding-004[12].
- **Optimization:** Embeddings are pre-computed and stored in a local NumPy array (shl_embeddings.npy) to ensure sub-second latency during inference, avoiding repeated API calls for static catalog data[15].

PAGE 2: Logic, Challenges & Evaluation

3. The Recommendation Logic

The core innovation of this solution is the "Balanced Retrieval" Algorithm[16]. A naive vector search often returns 10 variations of the same test (e.g., 10 different Java tests)[17]. However, a recruiter needs a diverse toolkit[18].

The Workflow:

1. **Input Processing:** The user's query is cleaned and embedded into a 768-dimensional vector[19].
2. **Cosine Similarity Search:** We calculate the cosine distance between the query vector and all 377 assessment vectors to find the top 20 semantic matches[20].
3. **The Balancing Step:**
   - The system tags results as either Technical or Behavioral based on metadata[21].
   - It re-ranks the top results to interleave these categories[22].
   - **Result:** Instead of 10 Java tests, the user gets ~7 Java tests and ~3 relevant Personality/Cognitive ability tests[23].
4. **Constraint Filtering:** If the user specifies "under 30 minutes," the system filters out long assessments before the final ranking[24].

4. Technical Challenges & Solutions

- **Challenge 1: Handling "Binary Garbage" in Data Input**
  - **Issue:** The provided test.csv file appeared to be an Excel file masked as a CSV, causing encoding errors (UnicodeDecodeError)[25].
  - **Solution:** I implemented a robust read_data_file function that automatically detects file signatures[26]. It attempts to read as standard CSV, then falls back to openpyxl (Excel reader), ensuring the pipeline never crashes due to bad user input[27].
- **Challenge 2: Cold Start Latency**
  - **Issue:** Generating embeddings for 300+ items on every startup is slow and costly[28].
  - **Solution:** Implemented a caching mechanism. The system checks for shl_embeddings.npy on boot. If found, it loads in 0.1 seconds; if missing, it triggers the build process[29].
- **Challenge 3: Deployment Consistency**
  - **Solution:** The entire application (API + Frontend) was Dockerized. A multi-stage Dockerfile installs dependencies and exposes the application on port 8000. This image was deployed to Render (Cloud), ensuring exact performance consistency[31].

5. Evaluation Methodology

To quantify the system's accuracy, I utilized the **Recall@K** metric, which measures how often the "correct" (Ground Truth) assessment appears in the top K recommendations[32].

- **Dataset:** train.csv (Provided by SHL)[33].
- **Metric:** Recall@10[34].
- **Results:** The system achieved a **Recall@10 score of ~1.0 (100%)** on the training data[35]. This indicates that for every sample query provided, our RAG engine successfully found the relevant assessment[36].