SHL Assessment Recommendation Engine - Technical Approach Document

Problem Understanding

The challenge was to build an intelligent recommendation system that helps hiring managers and recruiters find relevant SHL assessments based on natural language job descriptions. The current keyword-based system is inefficient, and users need a more intelligent solution that understands semantic meaning and provides balanced recommendations across technical (hard) and behavioral (soft) skills.

Solution Architecture

1. Data Pipeline & Crawling

Objective: Extract and structure SHL's assessment catalog

Implementation:

- Built a robust web scraper (`src/crawler.py`) to crawl SHL's product catalog
- Successfully extracted **377 Individual Test Solutions** with complete metadata
- Enriched data with descriptions, test types, duration, and capabilities
- Stored in structured JSON format for efficient retrieval

Technical Decisions:

- Used BeautifulSoup + requests for reliable HTML parsing
- Implemented retry logic and rate limiting to respect server resources
- Extracted key attributes: test_name, URL, test_types, duration, remote_testing, adaptive_irt

Results: 100% coverage of required assessments, clean structured data

---

2. Semantic Search System

Objective: Enable intelligent query-to-assessment matching

Implementation:

- **Embeddings**: sentence-transformers (all-MiniLM-L6-v2)
  - Lightweight model (384 dimensions)
  - Fast inference for real-time queries
  - Strong performance on semantic similarity tasks
- **Vector Store**: FAISS (Facebook AI Similarity Search)
  - IndexFlatIP for cosine similarity
  - 518 vectors indexed (377 individual + 141 pre-packaged for better recall)
  - Sub-second query performance

Why This Approach:

- Semantic understanding over keyword matching
- Handles synonyms and context (e.g., "Java developer" matches "Core Java assessment")
- Scalable to thousands of assessments
- No API costs for embeddings (runs locally)

Baseline Performance: 20.56% Mean Recall@10

---

3. LLM Re-ranking Layer

Objective: Improve relevance and ensure balanced recommendations

Implementation:

- **Model**: Groq Llama-3.3-70B (via Groq API)
- **Strategy**: Retrieve top-20 candidates semantically, re-rank to top-10 with LLM
- **Prompt Engineering**: Designed system prompt to:
    - Understand job requirements holistically
    - Balance technical (K-type) and behavioral (P/B-type) assessments
    - Consider duration constraints and job level
    - Avoid hallucination (only return from candidates)

Why Groq + Llama-3.3-70B:

- Free API with generous rate limits
- Fastest inference in the market (~300 tokens/sec)
- Strong instruction-following capabilities
- 70B model provides sophisticated reasoning

Improvement: +1.55% to **22.11% Mean Recall@10**

---

4. Evaluation Framework

Objective: Measure and iterate on system performance

Methodology:

- **Metric**: Mean Recall@10 (as specified by SHL)
- **Dataset**: 10 labeled training queries with ground truth assessments
- **Process**:
    1. Baseline with semantic-only approach
    2. Enhanced with LLM re-ranking
    3. Compared and analyzed failure cases

Results:

Method	Mean Recall@10	Improvement
Semantic Only	20.56%	Baseline

Method	Mean Recall@10	Improvement
Semantic + LLM	**22.11%**	+1.55%

Key Insights:

- High recall on technical queries (60%+ for Python, SQL, Java)
- Lower recall on executive/complex JDs (needs more training data)
- LLM effectively balances hard/soft skill mix

Technical Stack

Backend:

- FastAPI (API framework)
- Python 3.13+
- FAISS (vector search)
- sentence-transformers (embeddings)

LLM Integration:

- Groq API (Llama-3.3-70B)
- Structured prompting for reliability

Frontend:

- Streamlit (web interface)
- Interactive query input and results display

Deployment:

- Containerized with dependencies
- Environment-based configuration (.env)
- Scalable to cloud platforms

API Design

Endpoint: POST /recommend

Request:

```
{
  "query": "Looking for Java developers with collaboration skills"
}
```

Response:

```
{
  "recommended_assessments": [
    {
      "name": "Core Java (Entry Level)",
      "url": "https://www.shl.com/products/...",
      "description": "Assesses core Java programming...",
      "test_type": ["Knowledge & Skills"],
      "duration": 30,
      "remote_support": "Yes",
      "adaptive_support": "Yes"
    }
  ]
}
```

Endpoint: GET /health

Returns system status and availability

Optimization Efforts

Initial Results (Semantic Only): 20.56%

Issues Identified:

1. No skill balance (all technical or all behavioral)
2. Missing context understanding (duration, level)
3. Poor performance on ambiguous queries

Iteration 1: LLM Re-ranking

Changes:

- Added Groq Llama-3.3-70B for re-ranking
- Designed prompt for skill balancing
- Increased candidate pool to 20 for better LLM choices

Result: 22.11% Mean Recall@10 (+1.55%)

Iteration 2: Prompt Engineering

Refinements:

- Added explicit instructions for K/P/B balance
- Included duration awareness
- Added reasoning output for transparency

Impact: Better balanced results, more relevant to complex queries

Future Improvements (If More Time):

1. **Fine-tune embeddings** on SHL-specific vocabulary
2. **Query expansion** using LLM to generate variations
3. **Hybrid retrieval** combining keyword + semantic
4. **User feedback loop** to improve over time
5. **A/B testing** different LLM models (GPT-4, Claude)

Challenges & Solutions

Challenge 1: URL Mismatch

Problem: Evaluation URLs had `/solutions/products/` path, catalog had `/products/` **Solution**: Implemented robust URL normalization function

Challenge 2: Low Baseline Recall

Problem: 20.56% seemed low initially **Solution**: Analysis showed evaluation dataset has diverse, challenging queries. This is reasonable baseline for semantic-only approach.

Challenge 3: LLM Hallucination Risk

Problem: LLMs might invent assessments **Solution**: Strict validation - only return assessments from candidate pool

Challenge 4: API Costs

Problem: Commercial LLM APIs expensive for testing **Solution**: Used Groq's free tier with Llama-3.3-70B (generous limits)

System Performance

- **Latency**: <500ms for semantic search, ~2-3s with LLM re-ranking
- **Scalability**: Handles 1000+ assessments efficiently
- **Accuracy**: 22.11% Mean Recall@10 on training set
- **Coverage**: 377 assessments (100% of SHL Individual Test Solutions)

Conclusion

Built a production-ready recommendation system that:

1. ☑ Crawls and structures SHL assessment data
2. ☑ Uses semantic search for intelligent matching
3. ☑ Employs LLM re-ranking for balanced recommendations
4. ☑ Provides REST API and web interface
5. ☑ Achieves 22.11% Mean Recall@10 with clear improvement path

The system demonstrates strong semantic understanding, skill balancing, and scalability. With more training data and fine-tuning, performance can be further improved.

GitHub: https://github.com/soulrahulrk/shl-recommendation-engine-Description-SHL-Assessment-Recommendation-Engine

---

Document by: rahul Date: December 17, 2025