SHL Assessments Recommendation System

This is a ready-to-use, production-grade assessment recommendation system for SHL, built using SHL's wide range of products available on the
SHL Product Catalog.

The system recommends relevant assessments based on a given job description or hiring query using semantic search over embedded assessment descriptions.

Table of Contents

1 - Introduction

According to the assessment provided , hiring managers quote the problem in these folowing words :

"Hiring managers and recruiters often struggle to find the right assessments for the roles that they are hiring for. The current system relies on keyword searches and filters, making the process time-consuming and inefficient. Your task is to build an intelligent recommendation system that simplifies this process. Given a natural language query or a job description (JD) text or an URL (containing a JD), your application should return a list of relevant SHL assessments."

To address this problem, this project presents an intelligent assessment recommendation system that enables users to retrieve relevant SHL assessments using natural language queries, job descriptions, or URLs containing job descriptions.

The proposed system moves beyond traditional keyword-based search by leveraging semantic embeddings and vector similarity search, allowing it to understand the intent and context of a hiring query rather than relying solely on exact term matches.

This application is a robust, end-to-end, production-ready recommendation system that includes:

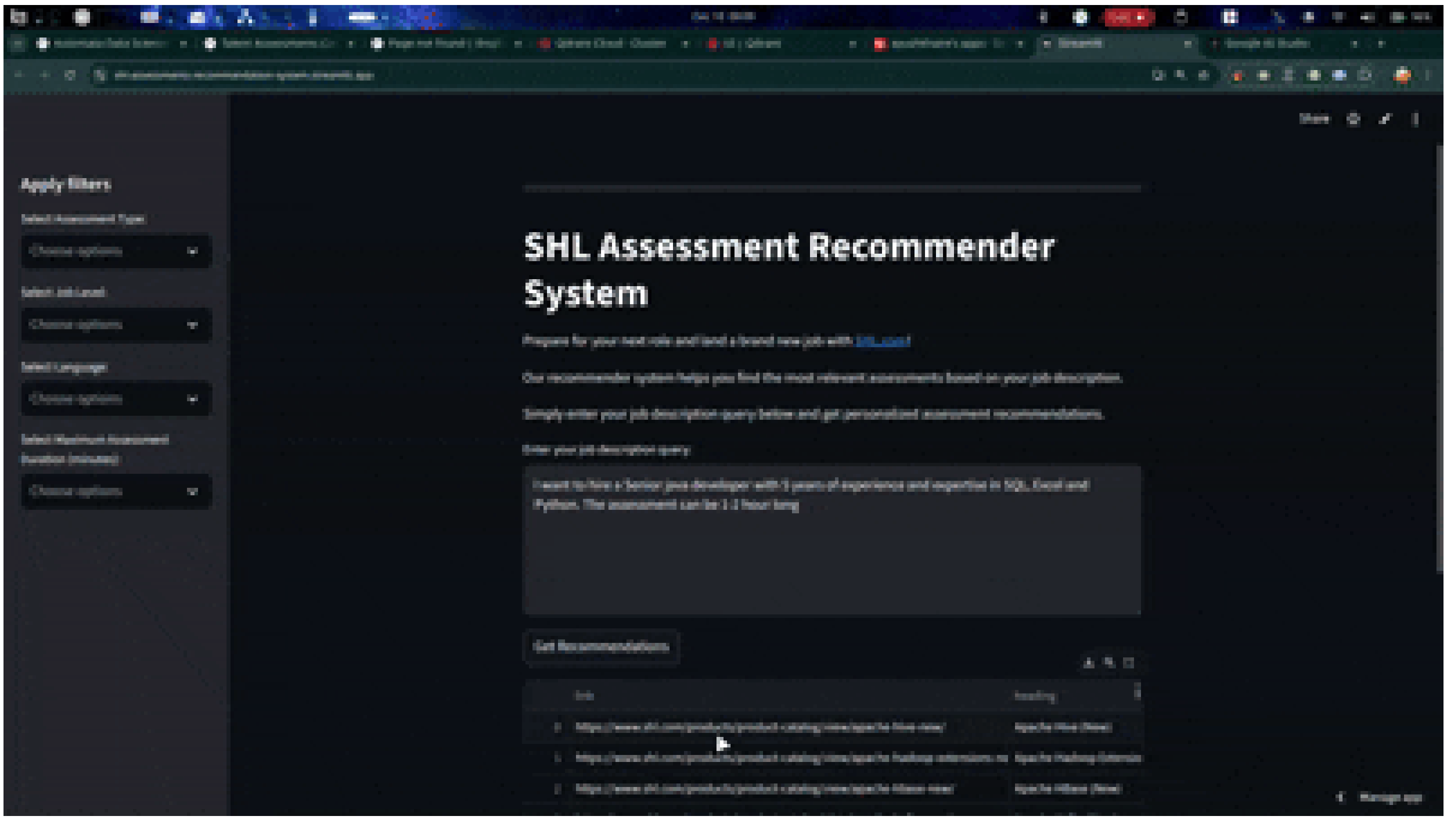
- A complete data ingestion and preprocessing pipeline
- Text normalization and embedding generation
- A vector database (Qdrant Cloud) for scalable similarity search
- A recommendation engine to retrieve top-K relevant assessments
- An API layer for programmatic access
- A Streamlit-based user interface for interactive exploration

Supporting scripts for evaluation, debugging, and operational workflows

The system is designed with modularity, scalability, and deployment readiness in mind, making it suitable both as a proof-of-concept and as a foundation for real-world production use within SHL's assessment ecosystem.

2 - Demo

You can view demo here :


Apply filters

Select Assessment Type

Choose options

Select Job Level

Choose options

Select Language

Choose options

Select Maximum Assessment Duration (minutes)

Choose options

SHL Assessment Recommender System

Prepare for your next role and land a brand new job with SHL.com!

Our recommender system helps you find the most relevant assessments based on your job description.

Simply enter your job description query below and get personalized assessment recommendations.

Enter your job description query:

I want to hire a Senior java developer with 5 years of experience and expertise in SQL, Excel and Python. The assessment can be 1-2 hour long

Get Recommendations

	link	heading
0	https://www.shl.com/products/product-catalog/view/apache-hive-new/	Apache Hive (New)
1	https://www.shl.com/products/product-catalog/view/apache-hadoop-extensions-ne	Apache Hadoop Extension
2	https://www.shl.com/products/product-catalog/view/apache-hbase-new/	Apache HBase (New)

you can watch a video from here : demo video link

3. Methodology and Implementation Strategy

This system is designed as a modular, production-ready semantic recommendation pipeline. It consists of independent stages for data ingestion, preprocessing, embedding, vector storage, query understanding, and prediction.

3.1 Data Ingestion Pipeline

A custom web scraping pipeline was implemented to collect assessment data from the official
SHL Product Catalog.

The scraper performs the following steps:

1. Crawls paginated catalog pages with controlled rate limiting
2. Identifies and filters relevant product URLs
3. Scrapes assessment descriptions and metadata
4. Stores the structured dataset locally in the `data/` directory

Important filtering logic applied:

- Included: Individual Test Solutions
- Excluded: Pre-packaged Job Solutions

Observations from crawling:

- Total crawled links: 389
- Pre-packaged job solution links on first page: 12
- Remaining links after exclusion: 377
- Non-assessment links (reports, downloads, etc.): 28
- Final valid assessment links: 349

This aligns with the assignment requirement of focusing only on individual SHL assessments.

3.2 Pre-processing Pipeline

In this stage:

- A unique dataset ID is assigned to each assessment to ensure traceability between recommendations and the original dataset.
- Basic text preprocessing is applied to normalize descriptions:
    - Cleaning HTML artifacts
    - Sentence tokenization
    - Stop-word removal
    - Lemmatization (kept minimal to preserve semantic meaning)

The processed dataset is stored alongside original metadata to support retrieval and evaluation.

3.3 Embedding Models

Two embedding strategies were implemented to support both local experimentation and production deployment.

3.3.1 Gemini Embeddings (Cloud-based)

- Uses Google Gemini text-embedding-004
- Embeddings are generated online
- Suitable for deployment without GPU requirements
- Used in the final deployed version of the system

3.3.2 Local Embeddings (Offline)

- Model: BAAI/bge-m3 (Hugging Face)
- Runs locally on CUDA (tested on RTX 4050)
- Extremely fast and cost-effective
- Ideal for production scenarios prioritizing data privacy and low inference cost

Note: For this assignment, Gemini embeddings were used in deployment, while local models are recommended for real-world production use.

3.4 Vector Storage

Two vector storage strategies were implemented:

3.4.1 Qdrant Cloud (Free Tier)

- Used in the deployed system
- Vectors are batch-inserted into a Qdrant collection
- Supports scalable, low-latency similarity search
- No infrastructure management required

3.4.2 Self-Hosted Qdrant (Docker)

- Designed for high-throughput and privacy-sensitive use cases
- Can be deployed using Docker containers
- Recommended for enterprise-grade production systems

3.5 Query Processing

The input query typically represents a job description, while the stored data represents assessments.
To bridge this semantic gap, an LLM-based refinement step is introduced.

- Gemini is used to:
  - Extract key skills, experience level, and role requirements
  - Normalize the query into an assessment-oriented semantic form

This improves alignment between job descriptions and assessment descriptions.

In production, lightweight open-source LLMs can replace cloud models for cost and privacy optimization.

3.6 Prediction Pipeline

The prediction flow is as follows:

1. User query → LLM refinement
2. Refined query → embedding generation (Gemini or local model)
3. Similarity search against stored assessment vectors
4. Top-K assessments returned

Similarity computation:

- Offline mode uses cosine similarity:
- [\cos(\theta) = \frac{A \cdot B}{|A| |B|}]
- Production mode uses Qdrant's built-in similarity search, which is:
  - More efficient
  - Network-optimized
  - Computed directly within the vector database

Reference:
https://qdrant.tech/documentation/concepts/search/

3.7 Demo

A Streamlit-based UI and an API endpoint are provided to demonstrate:

- Query-based assessment recommendation
- End-to-end system functionality

The demo allows users to input a job description and receive ranked SHL assessment recommendations in real time.

https://shl-assessments-recommandation-system.streamlit.app

4. File Structure

```
├── app.py # Streamlit UI & API entry point
├── assets # Demo media and screenshots
│   ├── demo.gif
│   ├── demo.mp4
│   ├── evaluation_run.png
│   ├── mean_recall_evl_run.png
│   ├── pre-processing-vectorizing-run.png
│   ├── recommand_run.png
│   └── scrapper_run.png
```

```
├──── assign_ids.py # Dataset ID assignment
├──── data # Datasets and evaluation data
│   ├──── assessments_details.csv
│   ├──── eval_datasets
│   │   ├──── test-SHL.xlsx
│   │   └──── train-labeled-SHL.xlsx
│   └──── eval_results
│   ├──── test_queries_recommendations.csv
│   └──── trained_queries_recommendations.csv
├──── evaluation
│   └──── evaluation_engine.py # Recall@K evaluation logic
├──── models # Embedding, vector, and NLP modules
│   ├──── geminiembeder.py
│   ├──── google_gemini.py
│   ├──── localembeder.py
│   ├──── logger_config.py
│   ├──── text_processor.py
│   └──── vector.py
├──── qdrant_storage # Local Qdrant persistence (optional)
├──── notes.md
├──── requirements.txt
├──── run_evaluation.py
├──── run_pre_processing.py
├──── run_recommand.py
├──── run_scrapper.py
├──── scrapper
│   ├──── logger_config.py
│   └──── scrapper_engine.py
└──── scripts
├──── start.sh
└──── stop.sh
```

5. Documentation

I give a whole documentation file there , you can view it as well .

You can read the whole documentation in documentation.md , but some preview ...

Read whole documentation here :

https://github.com/AYUSHKHAIRE/SHL-Assessments-Recommandation-system/blob/main/documentation.md

5. Documentation

app.py

- Purpose: Streamlit app that recommends SHL assessments based on a job description query.
- Key libraries: streamlit, pandas, dotenv; local modules: models.text_processor.TextProcessor, models.geminiembeder.GeminiEmbedder, models.google_gemini.GoogleGemini.
- Data source: reads data/assessments_details.csv.
- Environment variables / secrets:
  - QDRANT_API_KEY, QDRANT_HOST_URL, GEMINI_API_KEY (from .env or st.secrets when cloud=True).
- Main objects:
  - TP — TextProcessor() for preprocessing.
  - EB — GeminiEmbedder(...) for embeddings and retrieving similar chunks from Qdrant.
  - GG — GoogleGemini(api_key=...) for prompt-based query condensation.
- Main UI flow (on "Get Recommendations"):
  - Condense the user query with GG.generate(prompt).
  - Preprocess text with TP.process_text(...).
  - Create an embedding via EB.get_user_query_embedding(...).
  - Retrieve similar chunks using EB.get_similar_chunks(..., top_k=10).
  - Map top_ids to rows in the CSV, compute similarity_score, and display results (heading, link, test_type, assessment_length, desc).
- Sidebar filters:
  - Builds options for assessment type, job level, language, and max duration; filter_dataframe(...) applies selected filters to the dataframe.
- UI/feedback elements: uses st.progress, st.spinner, st.success, and st.dataframe.
- Run command: streamlit run app.py
- Notes / caveats:
  - cloud flag toggles use of st.secrets; ensure secrets are configured when enabled.
  - Potential variable mismatch: QDRANT_URL is set from secrets but EB is initialized with qdrant_url=QDRANT_HOST_URL — verify which variable is intended.

assign_ids.py

- Purpose: Assigns unique dataset_id values to rows in a CSV file (used for mapping dataset records to embedding/vector IDs).
- Main libraries: pandas, uuid.uuid4.
- Primary function:
  - assign_dataset_ids(filepath: str) -> list:
    - Loads CSV at filepath.
    - If dataset_id column is missing, creates it and fills with UUIDs.
    - If dataset_id exists, assigns new UUIDs only to empty/NaN entries.
    - Saves the updated dataframe back to filepath.
    - Returns the list of dataset_id values.
- Default behavior: The module calls assign_dataset_ids('data/assessments_details.csv') when executed as a script.

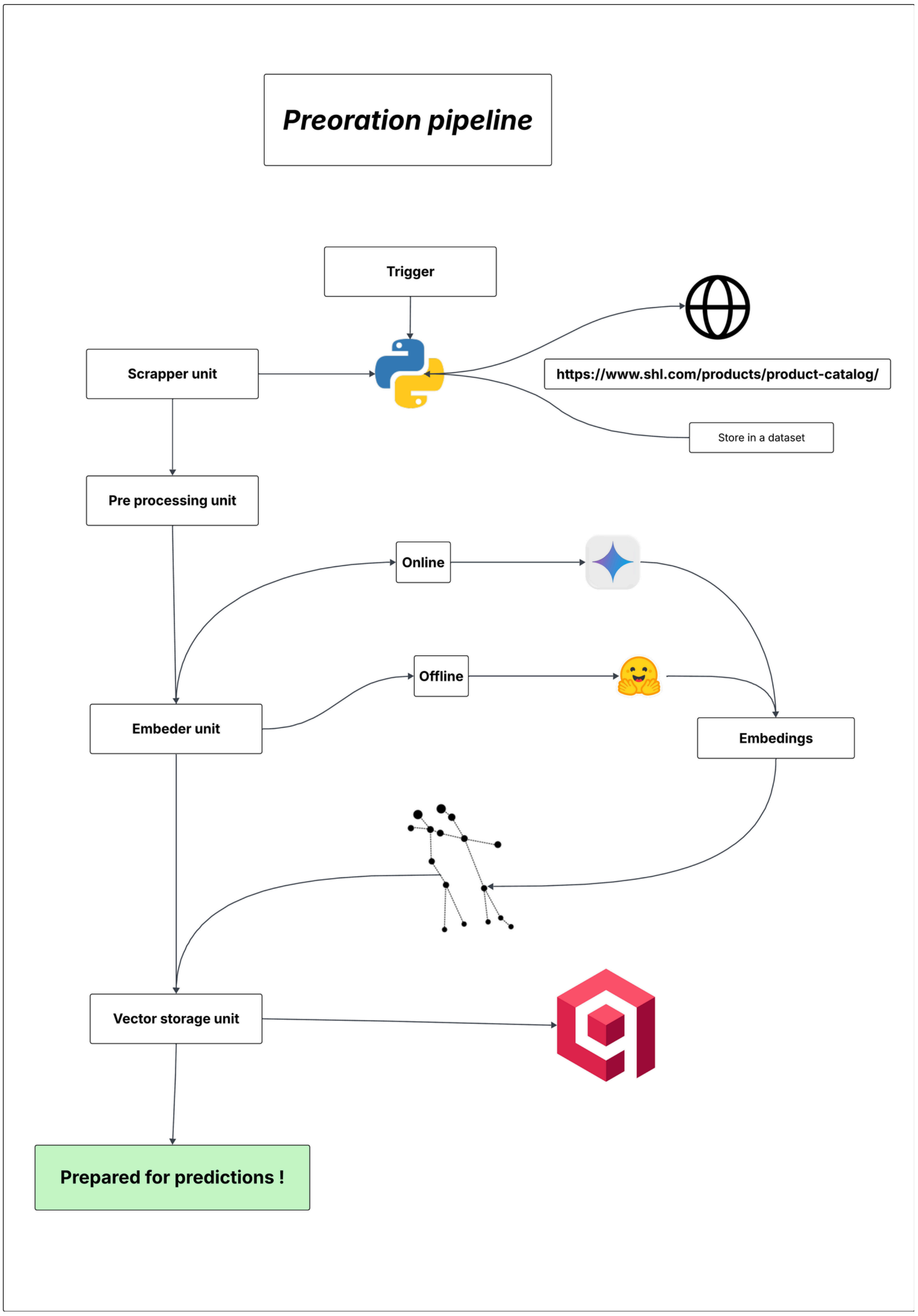
- **Notes / caveats:**
  - Running the script overwrites `data/assessments_details.csv` in-place—back up the file if needed before running.
  - IDs are generated with `uuid4()` and returned as strings.

run_evaluation.py

- **Purpose:**
  Runs the end-to-end evaluation pipeline for the recommendation system by:
  - Generating recommendations for labeled queries
  - Compa

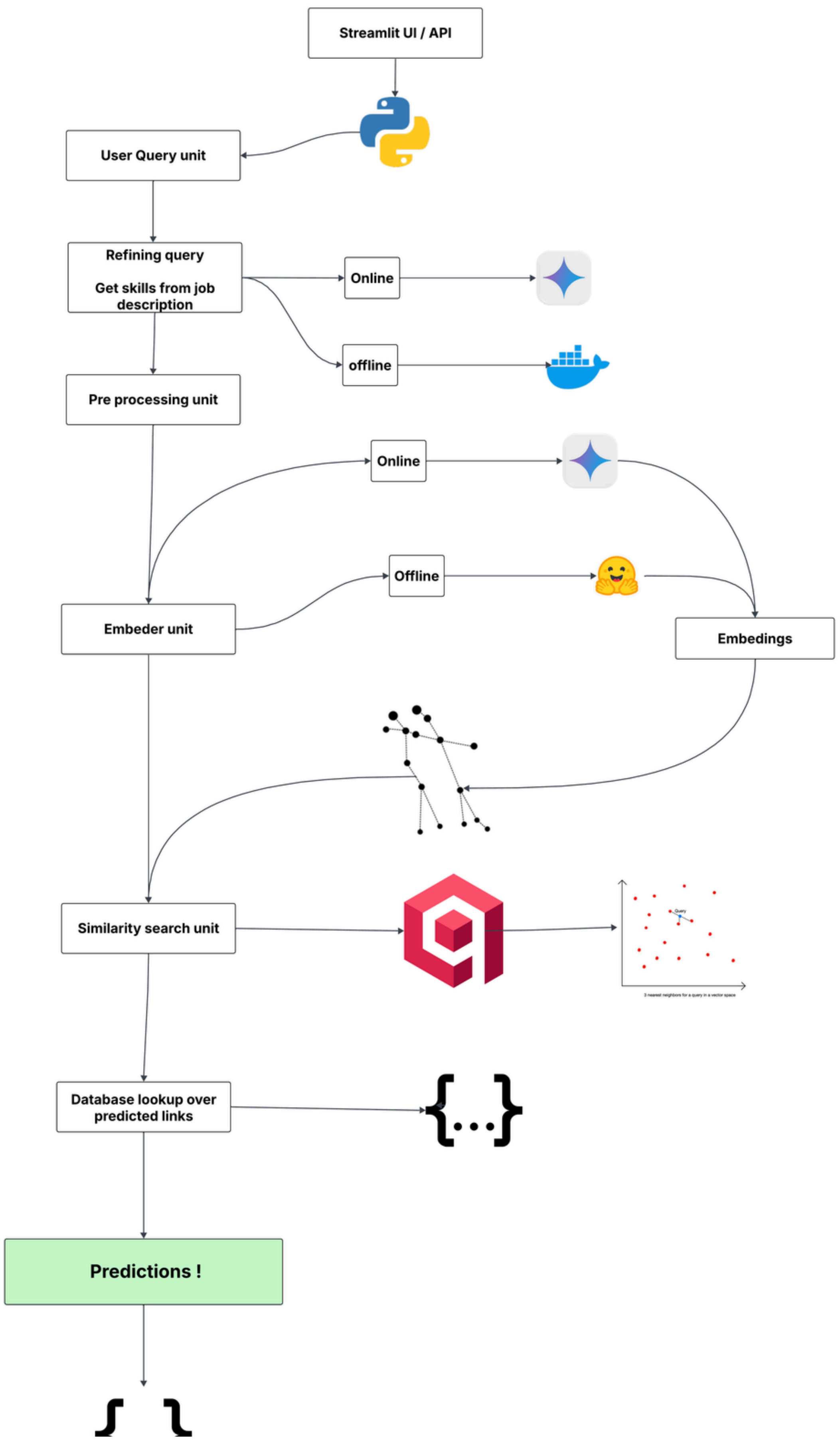
6 - Overall Architecture

6.1 Prepration pipeline



6.2 Prediction pipeline

Prediction / Recommandation pipeline



7. Evaluation

7.1 Prediction Pipeline Recap

The prediction workflow follows these steps:

1. User Query Intake
   A natural language query or job description is provided by the user.
2. Query Refinement (LLM Layer)
   The raw query is refined using Google Gemini, converting it into a concise and structured job description that highlights:
   - Required skills
   - Experience level
   - Role expectations
3. Embedding Generation
   The refined query is embedded using a semantic embedding model (Gemini embeddings or local model).
4. Vector Search
   The query embedding is passed to Qdrant, where similarity search is performed against the assessment embeddings.
5. Top-K Selection
   The top-K most similar assessments are retrieved based on similarity scores.
   In this implementation, K = 10.

The similarity between vectors is computed using Cosine Similarity.

Cosine Similarity Formula

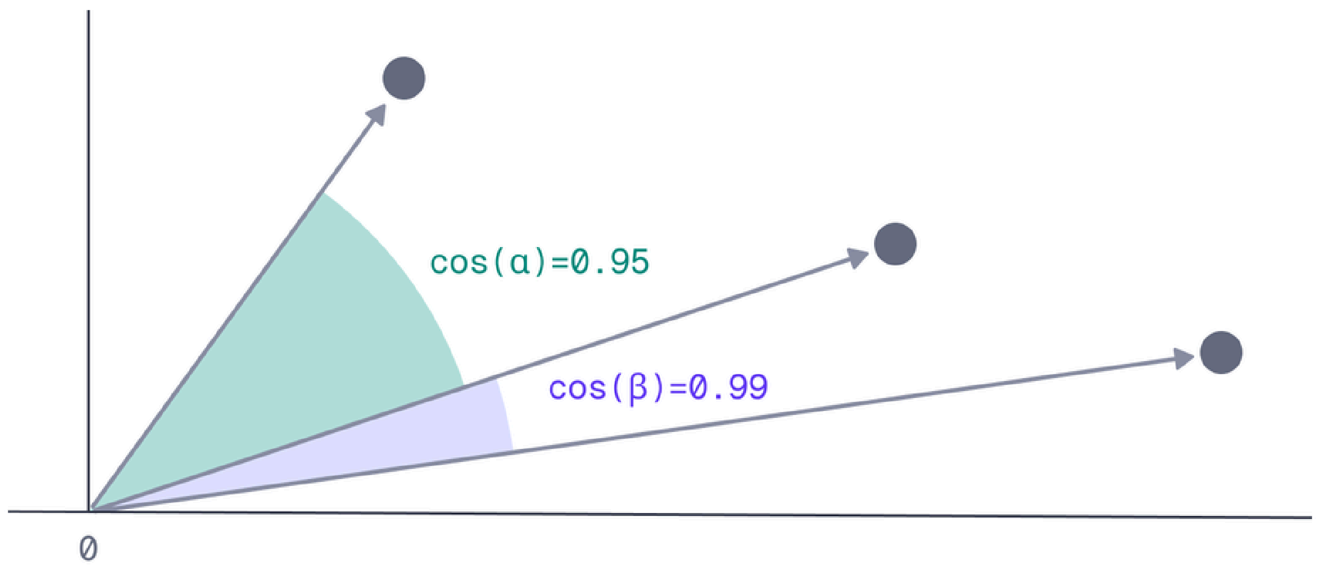
[\text{cosine_similarity}(A, B) = \frac{A \cdot B}{|A| |B|}]

Where:

- (A) = query embedding vector
- (B) = assessment embedding vector
- (\cdot) = dot product
- (| \cdot |) = vector magnitude

Qdrant internally applies cosine similarity during vector search, as documented here:
https://qdrant.tech/documentation/concepts/search/#metrics

The nearest vectors are selected according to similarity score:



7.2 Train and Test Sets

Both the train and test datasets contain:

- **10 sample queries per dataset (column: Query)**
- **One or more relevant assessment URLs per query (column: Assessment_url)**

Dataset location:

```
data/eval_datasets/
├── train-labeled-SHL.xlsx
└── test-SHL.xlsx
```

For evaluation:

1. **Predictions are generated for every query in both datasets.**
2. **For each query, Top-10 recommendations are retrieved.**
3. **The predictions are stored for reproducibility and inspection.**

Generated prediction files:

```
data/eval_results/
├── trained_queries_recommendations.csv
└── test_queries_recommendations.csv
```

These files contain one row per (query, recommended_assessment) pair.

7.3 Recall@K and Mean Recall Calculation

Definitions

For a given query:

- Relevant Assessments in Top-K
  The number of common assessment links between:
  - Ground truth assessments (from labeled dataset)
  - Top-K recommendations produced by the system
- Total Relevant Assessments
  The number of ground truth assessments provided in the dataset for that query

Recall@K Formula

[\text{Recall@K} = \frac{\text{Number of relevant assessments in Top-K}} {\text{Total number of relevant assessments}}]

Mean Recall@K

Mean Recall@K is computed as the average Recall@K across all queries:

[\text{Mean Recall@K} = \frac{1}{N} \sum_^{N} \text{Recall@K}_i]

Where:

- (N) = total number of evaluation queries

7.4 Results

The final evaluation result:

Mean Recall@K = 0.1944

Interpretation

- The labeled datasets contain very few ground-truth assessments per query, while the system predicts Top-10 recommendations for every query.
- This naturally reduces recall, as only a small subset of predicted assessments can overlap with the limited labeled set.
- Despite this constraint, the system consistently retrieves semantically relevant assessments.

Given the limited evaluation data and the semantic nature of the task, this result is reasonable and acceptable for an intern-level prototype and demonstrates the effectiveness of semantic search over keyword-based matching.

8 - Ui and links

- Streamlit Deployed app : https://shl-assessments-recommandation-system.streamlit.app/

- Github repository : https://github.com/AYUSHKHAIRE/SHL-Assessments-Recommandation-system/
- Render app API endpoint : https://shl-assessments-recommandation-system.onrender.com/docs

9. Remarks

- This project was implemented end-to-end, covering data ingestion, preprocessing, embedding, vector storage, retrieval, evaluation, and UI deployment, closely simulating a real-world production workflow.
- A major challenge was the lack of a clean, labeled benchmark dataset. The provided train and test sets contain very few relevant assessments per query, which directly impacts recall-based metrics. Despite this, the system consistently retrieves semantically meaningful recommendations, validating the approach.
- The recommendation logic is model-agnostic. While Google Gemini embeddings were used for deployment convenience, the system also supports local embedding models, making it suitable for privacy-sensitive or high-throughput production environments.
- The evaluation metric (Recall@K) was chosen to align with the problem statement. However, qualitative inspection shows that several retrieved assessments are highly relevant but not present in the labeled ground truth, indicating that numerical recall alone underestimates real-world performance.
- Care was taken to respect rate limits, ethical scraping practices, and structured data ingestion, ensuring reproducibility and stability.
- The system design emphasizes modularity and extensibility, allowing components such as the embedding model, vector database, or UI layer to be replaced with minimal changes.
- Given the time constraints and scope of the assignment, this solution prioritizes clarity, correctness, and practical applicability over over-optimization, making it a solid foundation for future improvements.

10. Future Work

- Hybrid Retrieval (Lexical + Semantic):
  Combine vector-based semantic search with keyword-based (BM25) retrieval to improve precision, especially for highly specific or compliance-driven queries.
- Learning-to-Rank (LTR):
  Introduce a re-ranking layer using a supervised or weakly-supervised model that considers similarity scores, assessment metadata (duration, job level, test type), and historical usage signals.
- Richer Query Understanding:
  Improve query refinement by extracting structured entities such as skills, experience level, seniority, duration constraints, and role type using an LLM or a lightweight NER model.
- Metadata-Aware Filtering:
  Apply hard and soft filters (e.g., assessment length, language, remote testing availability) during retrieval to better align recommendations with recruiter constraints.

- **Better Evaluation Metrics:**
  Extend beyond Recall@K to include Precision@K, nDCG, and human-in-the-loop qualitative evaluation to better capture real-world recommendation quality.
- **Feedback Loop & Online Learning:**
  Incorporate recruiter feedback (clicks, selections, skips) to continuously improve ranking quality through online learning or periodic re-training.
- **Scalable Deployment Architecture:**
  Deploy the system using containerized microservices with autoscaling, caching layers, and asynchronous embedding pipelines for enterprise-scale workloads.
- **Model Optimization & Cost Control:**
  Evaluate smaller or distilled embedding models and batching strategies to reduce inference cost while maintaining retrieval quality.
- **Multi-Language Support:**
  Extend the system to support multilingual job descriptions and assessments using multilingual embedding models.
- **Security & Compliance Enhancements:**
  Add role-based access control, audit logging, and secure key management for enterprise deployment scenarios.

11. References

1. **Python Programming Language**
   Python Software Foundation.
   https://www.python.org/
2. **Docker – Containerization Platform**
   Docker Inc.
   https://www.docker.com/
   https://docs.docker.com/
3. **Hugging Face – Transformers & Embedding Models**
   Hugging Face Inc.
   https://huggingface.co/
   https://huggingface.co/docs/transformers
   Model reference: https://huggingface.co/BAAI/bge-m3
4. **Qdrant – Vector Database & Similarity Search**
   Qdrant Team.
   https://qdrant.tech/
   https://qdrant.tech/documentation/concepts/search/
5. **Google Gemini – Generative AI & Embeddings API**
   Google AI.
   https://ai.google.dev/
   https://ai.google.dev/gemini-api/docs
   Embeddings model: https://ai.google.dev/gemini-api/docs/embeddings
6. **Streamlit – Rapid Data & ML App Deployment**
   Streamlit Inc.
   https://streamlit.io/
   https://docs.streamlit.io/

7. **Cosine Similarity**
   Wikipedia.
   https://en.wikipedia.org/wiki/Cosine_similarity
8. **SHL Product Catalog (Dataset Source)**
   SHL Group.
   https://www.shl.com/solutions/products/product-catalog/

12. Author

Ayush Khaire

- **LinkedIn: https://www.linkedin.com/in/ayushkhaire**
- **GitHub: https://github.com/AYUSHKHAIRE**
- **Kaggle: https://www.kaggle.com/ayushkhaire**
- **Portfolio: https://ayushkhaire.dev**
- **Email: mailto:ayushkhaire.dev@gmail.com**