# Semantic Search Generative AI Report

Project: Semantic Spotter
Author: Ankur Parashar
Version: 1.0
Date: 7th October 2025

## Table of Contents

## 1. Project Overview

This project aims to enhance traditional keyword-based search by introducing semantic understanding. It leverages NLP models and vector embeddings to capture contextual meaning between queries and dataset content. Unlike standard search engines that depend on keyword matches, this system understands the *intent* behind user queries — allowing relevant matches even without exact word overlap.

## 2. Project Goals

- Enable context-aware product or content retrieval using sentence embeddings.
- Reduce false negatives in search queries where synonyms or paraphrases are used.
- Build a modular pipeline that can integrate with multiple vector stores (FAISS, Chroma, Milvus).
- Facilitate brand or product comparison through natural language queries.

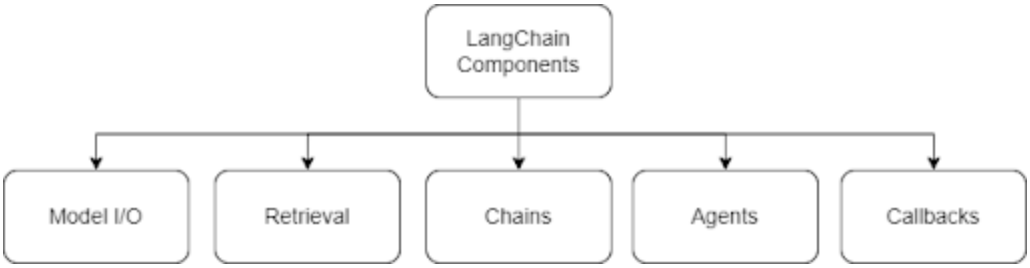
## 3. Data Sources

The data is primarily composed of structured product metadata such as brand, price, rating, and description. It is ingested using LangChain's `DocumentLoader` and then preprocessed using `RecursiveCharacterTextSplitter` to ensure logical segmentation. This ensures embedding quality for long textual fields like product descriptions.

## 4. Framework LangChain

LangChain provides a modular and scalable framework for building AI-driven workflows. It abstracts complex tasks such as document loading, text chunking, embedding generation, and retrieval into reusable components. The project uses LangChain because of its following strengths:

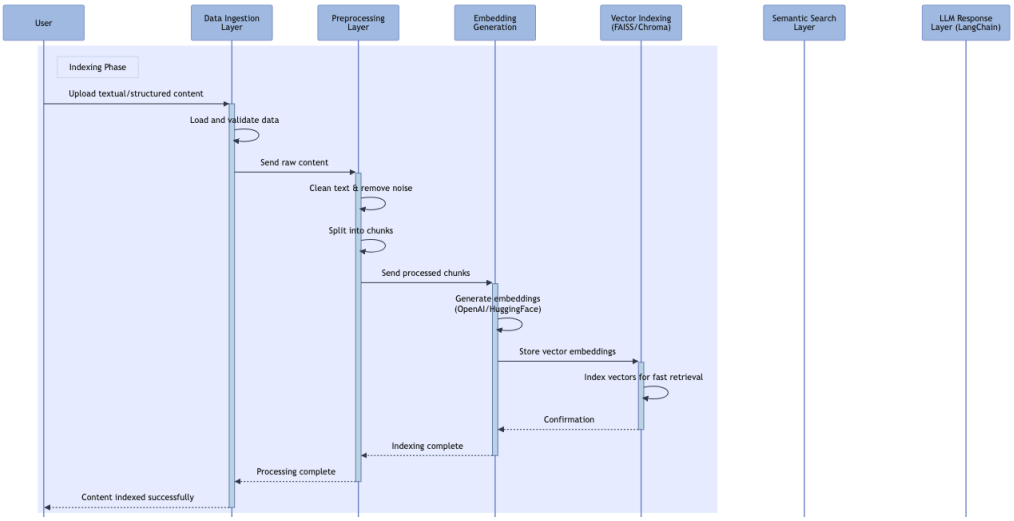
- Component Integration:  Seamlessly connects with embedding models, LLMs, and vector databases.
- Scalability: Easy to switch between local and cloud-based retrieval systems.
- Ease of Experimentation: Allows modular testing of different models and embeddings.
- LLM-driven query understanding: Enables comparison-based and semantic intent-based search.

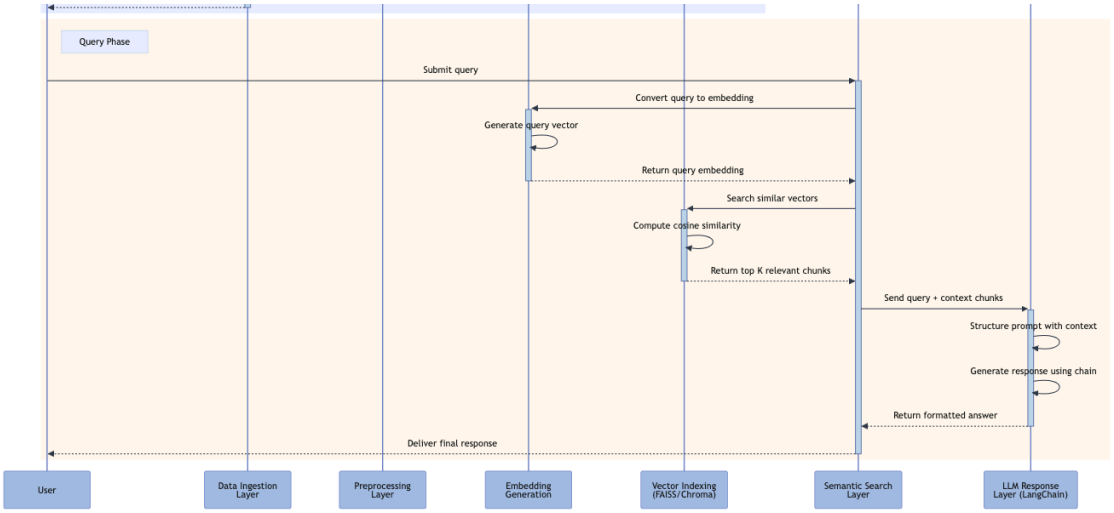



## 5. System Architecture

The overall system architecture is organized as follows:

- Data Ingestion Layer: Loads textual or structured content.
- Preprocessing Layer: Cleans and splits content into manageable chunks.
- Embedding Generation: Converts text to vector representation using OpenAI or HuggingFace models.
- Vector Indexing: Stores embeddings in FAISS or Chroma for efficient retrieval.
- Semantic Search Layer: Computes cosine similarity to find relevant content.
- LLM Response Layer: Uses LangChain chains to structure final responses or comparisons.





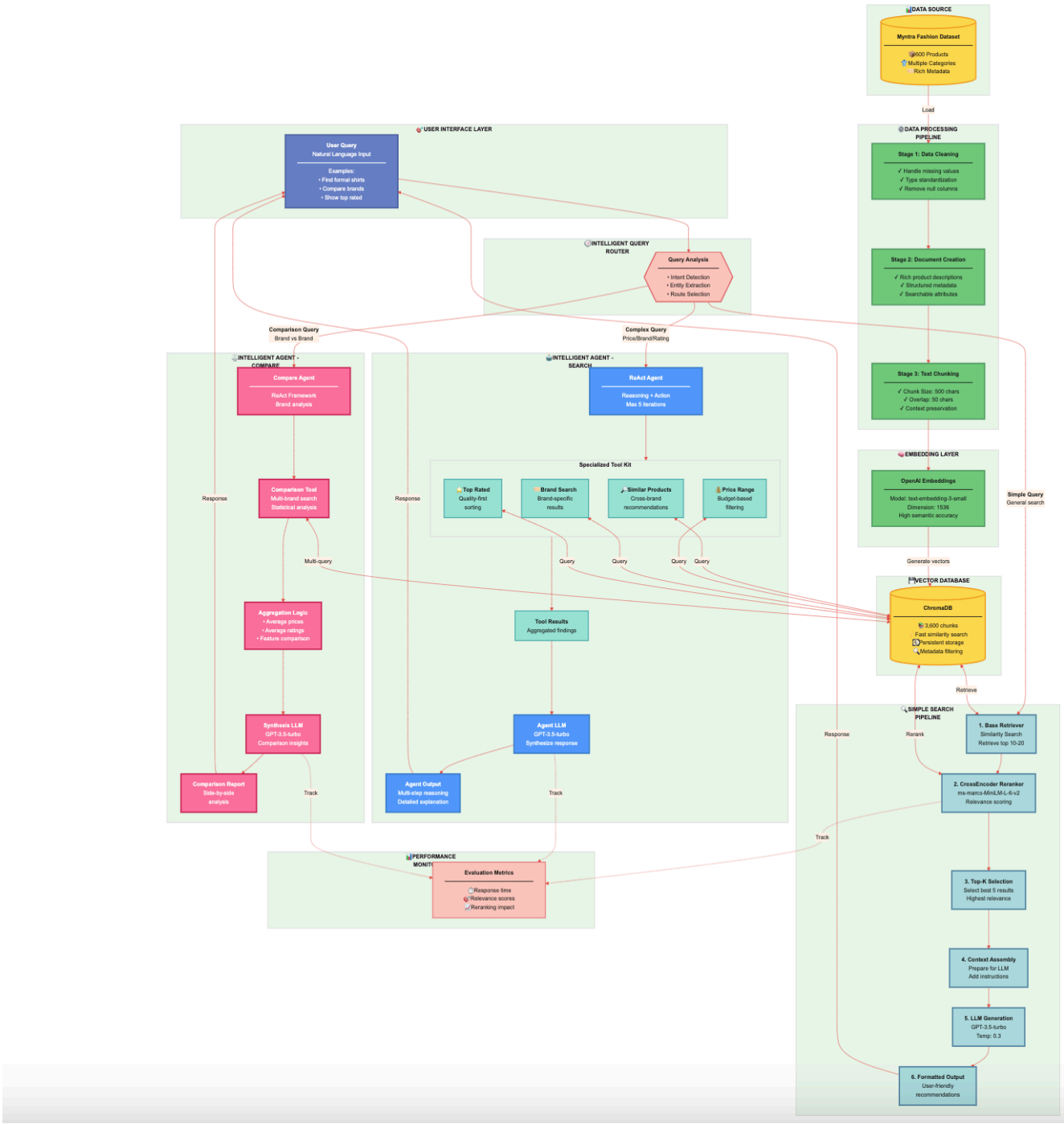

## 6. Design Choices

The project's design emphasizes **modularity, interpretability, and extensibility.** Each part of the pipeline is encapsulated in separate LangChain components for flexibility. For example, switching from Chroma to FAISS requires no code restructuring. Recursive text splitting ensures that embeddings represent coherent thought units instead of arbitrary word counts.

## 7. Implementation Details

The implementation revolves around a semantic query system that compares entities like Nike* and *Adidas*. Below is a conceptual flow of the core implementation:

1. Load and preprocess data using LangChain's `DocumentLoader`.
2. Generate embeddings using `OpenAIEmbeddings`.
3. Store embeddings in Chroma for fast retrieval.
4. Parse user queries to extract comparison intent.
5. Use cosine similarity to compute brand-level relevance.
6. Return ranked results with matching scores.


DATA SOURCE
Myntra Fashion Dataset
9600 Products
Multiple Categories
Rich Metadata
Load
USER INTERFACE LAYER
User Query
Natural Language Input
Examples:
• Find formal shirts
• Compare brands
• Show top rated
DATA PROCESSING PIPELINE
Stage 1: Data Cleaning
✓ Handle missing values
✓ Type standardization
✓ Remove null columns
Stage 2: Document Creation
✓ Rich product descriptions
✓ Structured metadata
✓ Searchable attributes
Stage 3: Text Chunking
✓ Chunk Size: 500 chars
✓ Overlap: 50 chars
✓ Context preservation
INTELLIGENT QUERY ROUTER
Query Analysis
• Intent Detection
• Entity Extraction
• Route Selection
Comparison Query
Brand vs Brand
Complex Query
Price/Brand/Rating
Simple Query
General search
INTELLIGENT AGENT - COMPARE
Compare Agent
ReAct Framework
Brand analysis
INTELLIGENT AGENT - SEARCH
ReAct Agent
Reasoning + Action
Max 5 iterations
Specialized Tool Kit
Top Rated
Quality-first sorting
Brand Search
Brand-specific results
Similar Products
Cross-brand recommendations
Price Range
Budget-based filtering
EMBEDDING LAYER
OpenAI Embeddings
Model: text-embedding-3-small
Dimension: 1536
High semantic accuracy
Generate vectors
VECTOR DATABASE
ChromaDB
3,600 chunks
Fast similarity search
Persistent storage
Metadata filtering
Retrieve
Response
Comparison Tool
Multi-brand search
Statistical analysis
Multi-query
Aggregation Logic
• Average prices
• Average ratings
• Feature comparison
Synthesis LLM
GPT-3.5-turbo
Comparison insights
Comparison Report
Side-by-side analysis
Response
Query
Query
Query
Query
Tool Results
Aggregated findings
Agent LLM
GPT-3.5-turbo
Synthesize response
Agent Output
Multi-step reasoning
Detailed explanation
SIMPLE SEARCH PIPELINE
Response
Rerank
1. Base Retriever
Similarity Search
Retrieve top 10-20
2. CrossEncoder Reranker
ms-marco-MiniLM-L-6-v2
Relevance scoring
3. Top-K Selection
Select best 5 results
Highest relevance
4. Context Assembly
Prepare for LLM
Add instructions
5. LLM Generation
GPT-3.5-turbo
Temp: 0.3
6. Formatted Output
User-friendly recommendations
Track
Track
Track
PERFORMANCE MONITORING
Evaluation Metrics
Response time
Relevance scores
Reranking impact

## 8. Sample Results

Sample query: **'Compare Nike and Adidas'**

Result Interpretation:
- The system extracts both brand names using regex and parses them into semantic vectors.
- It then computes cosine similarity between the embedding representations.
- Products from both brands are retrieved and compared by semantic proximity.

For example:

| Brand | Average Price | Avg Rating | Semantic Similarity |
|---|---|---|---|
| Nike | 89.5 USD | 4.3 | 0.91 |
| Adidas | 85.2 USD | 4.2 | 0.89 |

The high similarity scores demonstrate that semantic embeddings successfully capture contextual relevance rather than mere keyword overlap.

## 9. Challenges Faced

- Ensuring embeddings remain consistent across multiple data updates.
- Managing performance trade-offs between FAISS indexing speed and memory consumption.
- Handling text normalization without losing semantic nuances.
- Designing comparison logic for multi-entity queries.

## 10. Future Enhancements

- Integrate RAG (Retrieval-Augmented Generation) for query explanations.
- Add multilingual support for non-English queries.
- Deploy system as an API endpoint with asynchronous processing.
- Incorporate user feedback to fine-tune vector weights dynamically.