

# AI-Powered E-Commerce Semantic Search System

**RAG, Chunking, Embeddings, Re-ranking, LLM-as-Judge, and Modular Orchestration  
(Without LangChain)**

---

## 1. Project Objective

Design and implement a production-style AI-powered e-commerce semantic search system using:

- HTML + CSS (frontend)
- Flask or FastAPI (backend)
- FAISS (vector search)
- SentenceTransformers (embeddings)
- Cross-encoder re-ranking
- OpenAI (LLM + LLM-as-Judge)
- Fully modular orchestration layer
- No LangChain or orchestration frameworks

The system must demonstrate a complete Retrieval-Augmented Generation (RAG) pipeline with evaluation and monitoring.

---

## 2. Problem Statement

Traditional keyword-based search systems fail when users express intent semantically rather than through exact matches.

Example query:

“Comfortable running shoes under 5000”

The system should:

- Understand semantic intent
- Retrieve relevant products
- Apply filtering and re-ranking
- Generate grounded recommendations
- Evaluate its own output using an LLM judge
- Log evaluation metrics for monitoring

The objective is to simulate a real-world, production-grade semantic search system used in modern e-commerce platforms.

---

## 3. Functional Requirements

### 3.1 Frontend (HTML + CSS)

Create a clean search interface with:

- Search input field
- Search button
- Results section (ranked products)
- LLM-generated explanation section
- Evaluation score display panel

When a user submits a query:

- The system must return ranked products
  - Display a generated explanation
  - Display evaluation scores from the judge model
-

### 3.2 Backend (Flask or FastAPI)

Expose API endpoint:

POST /search

Request:

```
{  
  "query": "comfortable running shoes under 5000"  
}
```

Response:

```
{  
  "products": [...],  
  "llm_explanation": "...",  
  "evaluation_score": {  
    "relevance": 4,  
    "faithfulness": 5,  
    "completeness": 4  
  }  
}
```

Backend must orchestrate the entire pipeline in modular form.

---

## 4. Data Ingestion Pipeline

Load product catalog (JSON or CSV) with fields such as:

- product\_id
- name
- description
- price
- category
- specifications
- reviews (optional)

Students must design a reusable ingestion module.

---

## 5. Chunking Strategy

Implement:

- Chunk size: 300–500 tokens
- Overlap: 50 tokens

Chunk:

- Description
- Specifications
- Reviews

Each chunk must store metadata:

```
{  
  "product_id": 101,  
  "chunk_id": "101_3",  
  "text": "...",  
  "price": 4999,  
  "category": "shoes"  
}
```

Chunking must be implemented as a separate module.

---

## 6. Embedding Layer

Use:

SentenceTransformer('all-MiniLM-L6-v2')

- 384-dimensional embeddings
- Normalize vectors

- Persist embeddings
- Store in FAISS index

Index configuration:

`faiss.IndexFlatIP(384)`

Index must be saved to disk and reloaded on application startup.

---

## 7. Semantic Retrieval (ANN Search)

Pipeline:

1. Embed user query
2. Perform ANN search (Top 20)
3. Apply metadata filtering (price, category)
4. Return candidate product set

Metadata filtering must be implemented separately from vector retrieval.

---

## 8. Re-ranking Layer

Use cross-encoder model:

`cross-encoder/ms-marco-MiniLM-L-6-v2`

Re-rank top 20 candidates and return top 5.

Demonstrate:

- Before re-ranking order
- After re-ranking order

Re-ranking must be modular and replaceable.

---

## 9. RAG Context Assembly

Construct structured context using retrieved products:

- Product name
- Description
- Price
- Key features

Ensure:

- No hallucinated products
- Only retrieved products are included

---

## 10. LLM Recommendation Generation

Use OpenAI chat model to:

- Recommend products
- Explain reasoning
- Justify ranking
- Avoid hallucination

System prompt must explicitly constrain the model to retrieved products only.

---

## 11. LLM-as-Judge Evaluation

Implement a judge model that evaluates:

- Relevance
- Faithfulness
- Completeness

Judge must:

- Return structured JSON
- Use deterministic configuration
- Validate output schema
- Log evaluation results

Evaluation example:

```
{  
  "relevance": 4,  
  "faithfulness": 5,  
  "completeness": 3,  
  "overall_score": 4.0  
}
```

---

## 12. Logging and Monitoring

Store:

- Query
- Retrieved products
- Re-ranked products
- LLM explanation
- Judge scores
- Timestamp

- Model versions

Provide metrics endpoint:

GET /metrics

Return:

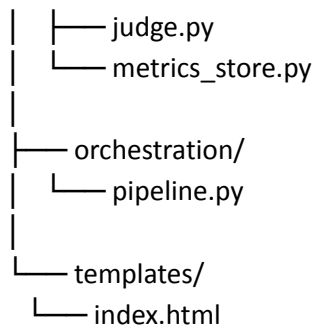
- Average relevance
  - Average faithfulness
  - Daily trend
  - Total queries evaluated
- 

## 13. Modular Architecture (Mandatory)

No monolithic script allowed.

Required project structure:

```
/project
|
|— app.py
|— config.py
|
|— ingestion/
|   |— loader.py
|   |— chunker.py
|   |— embedder.py
|
|— retrieval/
|   |— vector_store.py
|   |— retriever.py
|   |— reranker.py
|
|— generation/
|   |— prompt_builder.py
|   |— llm_generator.py
|
|— evaluation/
```



All modules must be reusable and independently testable.

---

## 14. Orchestration Layer

Implement a central pipeline class:

```
class EcommercePipeline:
    def __init__(self):
        self.retriever = Retriever()
        self.reranker = Reranker()
        self.generator = Generator()
        self.judge = Judge()

    def run(self, query):
        candidates = self.retriever.retrieve(query)
        ranked = self.reranker.rerank(query, candidates)
        response = self.generator.generate(query, ranked)
        score = self.judge.evaluate(query, ranked, response)
        return ranked, response, score
```

Pipeline must support:

- Turning judge on/off
  - Replacing reranker
  - Switching embedding model
  - Batch evaluation mode
- 

## 15. Production Constraints

Students must implement:

- FAISS index persistence
  - Embedding caching
  - Error handling
  - Timeout management
  - Secure API key management
  - Deterministic judge scoring
  - Structured logging
- 

## 16. Demonstration Requirements

During presentation, students must show:

1. Chunked data examples
  2. Embedding vector dimensions
  3. ANN retrieval output
  4. Re-ranking impact
  5. Final LLM response
  6. Judge scoring output
  7. Evaluation logs
- 

## 17. Success Criteria

The system must:

- Return semantically relevant results
  - Demonstrate re-ranking changes
  - Prevent hallucinated products
  - Display judge evaluation scores
  - Log evaluation metrics
  - Use fully modular architecture
  - Persist vector index
  - Demonstrate end-to-end orchestration
- 

## 18. Learning Outcomes

Students will gain practical understanding of:

- Retrieval-Augmented Generation
- Embedding pipelines
- ANN search systems
- Multi-stage retrieval
- Cross-encoder re-ranking
- Prompt engineering
- LLM-as-Judge evaluation
- Production monitoring
- Modular system design
- Forward Deployed Engineering mindset