Full-Scale LLM System Using RAG, Tool Calling, LCEL, Vector Databases, and OpenAI

---

1. Executive Problem Statement

Modern digital commerce platforms face increasing complexity in customer interactions. Users expect:

- Natural language product discovery
- Context-aware recommendations
- Real-time information
- Personalized suggestions
- Accurate product availability
- Intelligent fallback when products are unavailable

Traditional search systems rely on keyword matching and rule-based recommendation engines, which fail when:

- User queries are ambiguous
- Context (weather, trends, intent) matters
- Products are missing from the catalog
- Cross-domain reasoning is required

We propose building a full-scale AI-driven Commerce Assistant that integrates:

- Retrieval-Augmented Generation (RAG)
- Vector similarity search
- Tool calling for real-time external data

- Web fallback using Serper
- Weather-aware contextual recommendations
- OpenAI LLM reasoning
- Modular LCEL orchestration using LangChain
- Production-grade separation of concerns

This system is designed to be deployable in enterprise environments.

---

2. Objectives

The system must:

1. Provide semantic product search.
2. Deliver personalized recommendations.
3. Use contextual signals (e.g., weather).
4. Fallback to web search if products are unavailable.
5. Avoid hallucination via grounded RAG.
6. Maintain modular, scalable architecture.
7. Ensure deterministic tool execution.

---

3. Core Capabilities

3.1 Semantic Product Search

- User queries are converted into embeddings.

- Vector similarity search retrieves top-K relevant products.
- Metadata filtering applied (price, rating, availability).
- Retrieved documents passed to LLM for synthesis.

3.2 Context-Aware Recommendations

Example:
User query:
“Suggest jackets suitable for rainy weather in Mumbai under 5000.”

System flow:

1. Weather tool retrieves forecast.
2. Apply price filter.
3. Retrieve semantically relevant products.
4. Rank by weather suitability.
5. Generate grounded response.

3.3 Fallback Web Search

If:

- No product found
- Low similarity score
- Out-of-stock items

Then:

- Serper search tool is invoked

- External products summarized
- Clearly labeled as external sources

This ensures no dead-end user experience.

---

4. Architectural Principles

4.1 Separation of Concerns (SoC)

Layer	Responsibility
API Layer	Request handling
Orchestration Layer	Routing and state management
Retrieval Layer	Vector search
Tool Layer	External APIs
LLM Layer	Reasoning and synthesis
Data Layer	Product catalog and embeddings

Each layer is independently testable and replaceable.

4.2 Clean Architecture

- Business logic isolated from infrastructure
- LLM usage abstracted
- Tools registered centrally
- Vector DB independent of orchestration
- Retrieval modular and pluggable

---

5. Milestone 3 — System Architecture

5.1 High-Level Architecture

User
↓
API Gateway
↓
Orchestrator (LCEL Pipeline)
↓
Intent Classifier
↓
Router
├── RAG Pipeline
├── Recommendation Engine
├── Weather Tool
├── Serper Tool
└── Utility Tools
↓
LLM Synthesis
↓
Response

5.2 Intent Classification

An initial LLM pass determines:

- Product Search
- Recommendation
- Weather Query
- Financial Calculation
- General Knowledge

Routing is conditional and explicit.

---

6. Milestone 4 — RAG Design

6.1 Data Ingestion Pipeline

1. Load product catalog.
2. Normalize text fields.
3. Chunk descriptions using RecursiveCharacterTextSplitter.
4. Generate embeddings using OpenAI.
5. Store vectors in ChromaDB.
6. Attach metadata (price, category, rating, stock).

6.2 Chunking Strategy

- Chunk size: 500–800 characters
- Overlap: 100–200
- Preserve semantic boundaries

This balances recall and precision.

6.3 Embedding Strategy

- Model: OpenAI text-embedding-3-small
- Store dense vectors
- Attach metadata filters

6.4 Vector Database

ChromaDB:

- Stores embeddings
- Supports metadata filtering
- Enables similarity search
- Supports persistence

---

7. LCEL-Based Orchestration

Instead of legacy chain classes, use explicit LCEL pipelines.

Example logical structure:

```
{
 "context": retriever | format_docs,
 "question": passthrough
}
| prompt
| llm
| parser
```

This ensures:

- Transparency
- Composability
- Production safety

---

8. Tool Calling Design

Registered tools:

- get_weather
- web_search (Serper)
- calculate_interest
- inventory_check
- product_lookup

LLM decides which tool to call via structured function calling.

The orchestration loop:

1. LLM reasoning
2. Tool call returned
3. Backend executes tool
4. Tool result returned
5. LLM synthesizes final response

---

9. Hybrid Retrieval Strategy

When similarity score is low:

1. Invoke Serper tool.
2. Retrieve web results.
3. Summarize content.
4. Clearly indicate external source.

This reduces hallucination risk.

---

10. Personalization Layer

Optional advanced layer:

- Generate user embedding from purchase history.
- Compare with product embeddings.
- Re-rank results.

Improves long-term user satisfaction.

---

11. Observability and Monitoring

Track:

- Tool invocation frequency
- Retrieval confidence scores
- Token usage
- Latency per component
- Fallback usage rate
- Failure rates

This is mandatory for enterprise deployment.

---

12. Safety and Guardrails

- Strict prompt grounding
- No unsupported product generation

- Explicit fallback to web
- Controlled tool invocation
- Metadata filtering to avoid hallucination

---

13. End-to-End Flow Example

User:
"Recommend waterproof hiking shoes under 4000 for Bangalore weather."

System:

1. Intent classified as recommendation.
2. Weather tool invoked.
3. Metadata filter price < 4000.
4. Vector retrieval.
5. Re-ranking based on weather suitability.
6. LLM generates grounded recommendation.
7. If none found → Serper fallback.

---

14. Technology Stack

Component	Technology
LLM	OpenAI GPT-4o-mini
Embeddings	OpenAI
VectorDB	Chroma
Framework	LangChain LCEL

Search	Serper
Weather	OpenWeather
API Layer	FastAPI
Deployment	Docker + Cloud

---

15. Why This Is Industry Grade

This design includes:

- Modular orchestration
- Deterministic tool execution
- Retrieval grounding
- Hybrid fallback strategy
- Separation of concerns
- Context-aware recommendations
- Explicit routing
- Extensibility
- Observability
- Production scalability