# LLM Application Architecture

---

## 1. Introduction

**LLM Application Architecture** defines how Large Language Models are **integrated into real software systems**. Unlike simple chatbots, production-grade LLM applications require:

- Clear separation of concerns
- Scalable design patterns
- Controlled data access
- Reliable conversation flows
- Business-aligned goals

---

## 2. Core Components of an LLM Application

- **Client (UI / API):** User interaction
- **Orchestrator:** Controls flow and logic
- **Prompt Layer:** Templates and constraints
- **Context Manager:** Maintains conversation state
- **LLM API:** Language model inference
- **Knowledge Source:** Documents, databases
- **Tooling / Actions:** APIs, functions
- **Observability:** Logging, metrics

---

## 3. Key LLM Design Patterns

---

## 4. Pattern 1: Retrieval-Augmented Generation (RAG)

### 4.1 What Is RAG?

**Retrieval-Augmented Generation (RAG)** combines LLM reasoning capabilities with external knowledge retrieval. Instead of relying solely on model training data, RAG injects **relevant documents at runtime**.

### 4.2 RAG Architecture

```
User Query → Query Embedding → Vector Search
→ Relevant Documents → Prompt Injection → LLM Response
```

### 4.3 When to Use RAG

- Internal documentation (Private data)
- Regulatory content (Accuracy)
- Frequently changing data (Freshness)
- Large knowledge bases (Scalability)

---

## 5. Pattern 2: Agent-Based Systems

### 5.1 What Is an Agent?

An **LLM agent** is an autonomous component that:

- Has a goal
- Can plan steps
- Uses tools
- Evaluates results

### 5.3 Types of Agents

- Single-agent: Simple workflows
- Multi-agent: Complex coordination
- Tool-using agent: API interaction
- Validator agent: Output checking

---

## 6. Pattern 3: Prompt-Chain / Workflow Pattern

Break a complex task into **deterministic steps**.

```
Step 1: Extract entities
Step 2: Validate data
Step 3: Generate response
```

**Use Case:** Data processing pipelines, Document analysis, Compliance checks

---

## 7. Pattern 4: Tool-Calling Architecture

LLM acts as a **controller**, invoking backend services.

```
LLM → get_customer_data()
LLM → calculate_risk()
LLM → generate_summary()
```

---

## 8. Common Architectural Mistakes

- Treating LLM as a database
- No retrieval grounding
- Overusing agents
- No conversation goals
- Ignoring cost and latency

---

## 9. Best Practices Summary

- Choose patterns based on business goals
- Keep LLM stateless
- Externalize memory and knowledge
- Use RAG for factual accuracy
- Add agents only when needed

---

## 10. Conclusion

LLM Application Architecture is the foundation for building **scalable, safe, and business-aligned AI systems**. By applying proven design patterns such as **Retrieval-Augmented Generation and Agent-based systems**, organizations can move from experimental chatbots to **production-ready AI applications**.