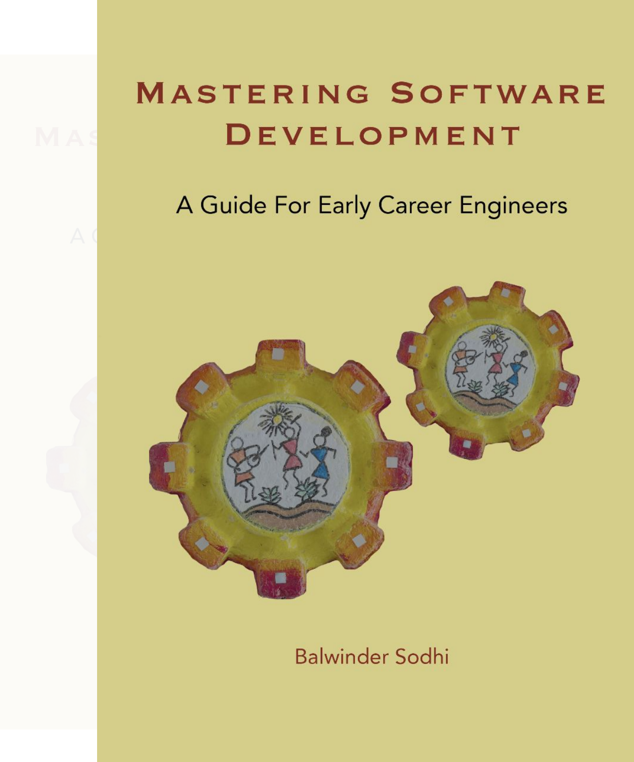
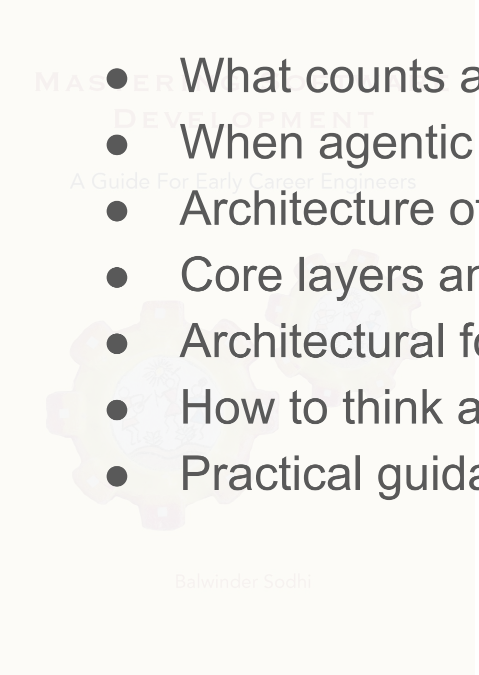

MASTERING SOFTWARE DEVELOPMENT
A Guide For Early Career Engineers
Balwinder Sodhi

LLM-powered and Agentic Applications

Balwinder Sodhi

What This Chapter Covers


Mastering Software Development
A Guide For Early Career Engineers
Balwinder Sodhi

- What counts as an LLM-powered application
- When agentic patterns are justified
- Architecture of agentic systems
- Core layers and interactions
- Architectural foundations and patterns
- How to think about trade-offs
- Practical guidance

What is an LLM-powered Application?

- A software system where LLM output meaningfully influences system behaviour.
- LLM is not the whole system; it’s an embedded computational capability.
- Typical use:
  - Text generation
  - Semantic retrieval
  - Classification / extraction
  - Reasoning and planning at small scale

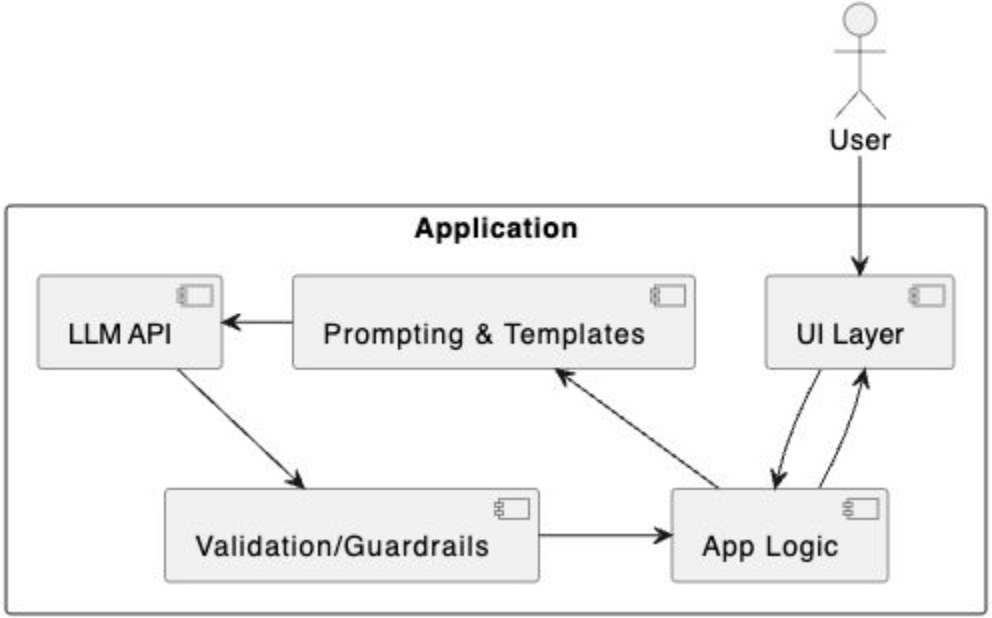
Anatomy of an LLM-Powered Application

- **App logic remains deterministic:** LLM augments it, not replaces it.
- Common components:
  - Prompting & templates
  - Vector search / RAG
  - Lightweight post-processing
  - Guardrails / validation
- Interaction pattern: **request → LLM → validate → integrate into workflow**

Mastering Software Development
A Guide For Early Career Engineers
Balwinder Sodhi

High-Level Structure of an LLM-Powered App

MASTERING SOFTWARE DEVELOPMENT
A Guide For Early Career Engineers
Balwinder Sodhi



Should You Build an Agentic Application or a Plain LLM-powered One?

You likely do not need agentic architectures when:

- The task is short-lived, stateless
- Input/output types are simple and well-defined
- No multi-step planning needed
- Errors are tolerable with retries
- Business logic is deterministic and bounded

Examples: summarization, classification, RAG chat, form filling, rewriting.

Mastering Software Development
A Guide For Early Career Engineers
Balwinder Sodhi

When You Should Consider Agentic Applications

Agentic architectures help when:

- You need multi-step workflows
- Actions depend on dynamic observation
- The system must self-correct
- External tools must be called in a feedback loop
- Long-running activities (minutes to hours)
- Goals cannot be achieved in one LLM call
- State is essential (context, memory, planning)

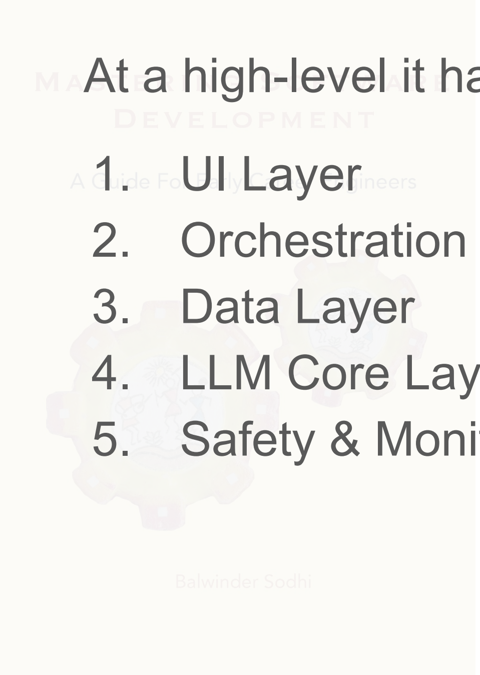
Examples: autonomous research agents, automated operations (ticket triage → fix), data pipeline agents, orchestration workflows.

DEVELOPMENT
A Guide For Early Career Engineers
Balwinder Sodhi

Comparing the Two

Dimension	LLM-Powered App	Agentic App
State	Mostly stateless	Maintains evolving state
Execution	Single request-response	Multi-step loop
Control	Developer-driven	Goal-driven with agent feedback
Failure handling	Retry	Plan revision, tool fallback
Complexity	Low	High
Best for	Assistive tasks	Autonomous workflows

Architecture of an Agentic Application

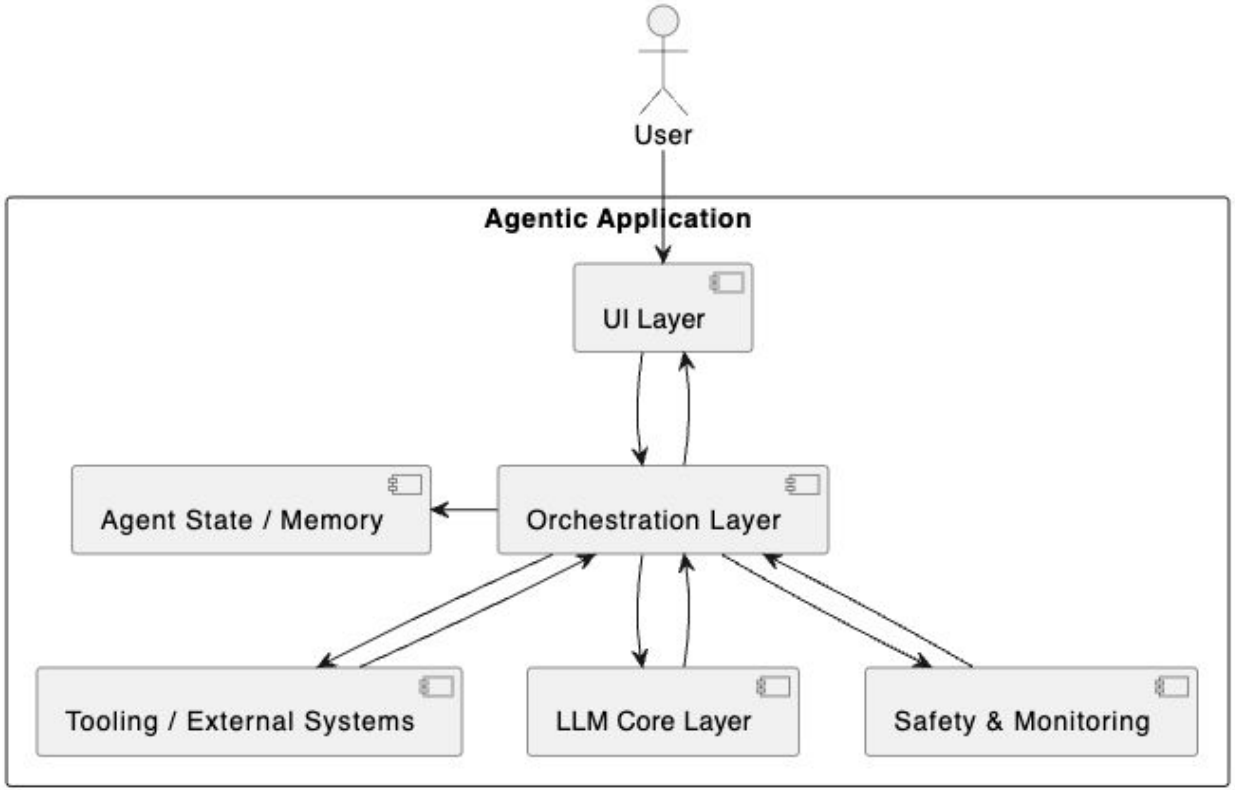


At a high-level it has five major layers:

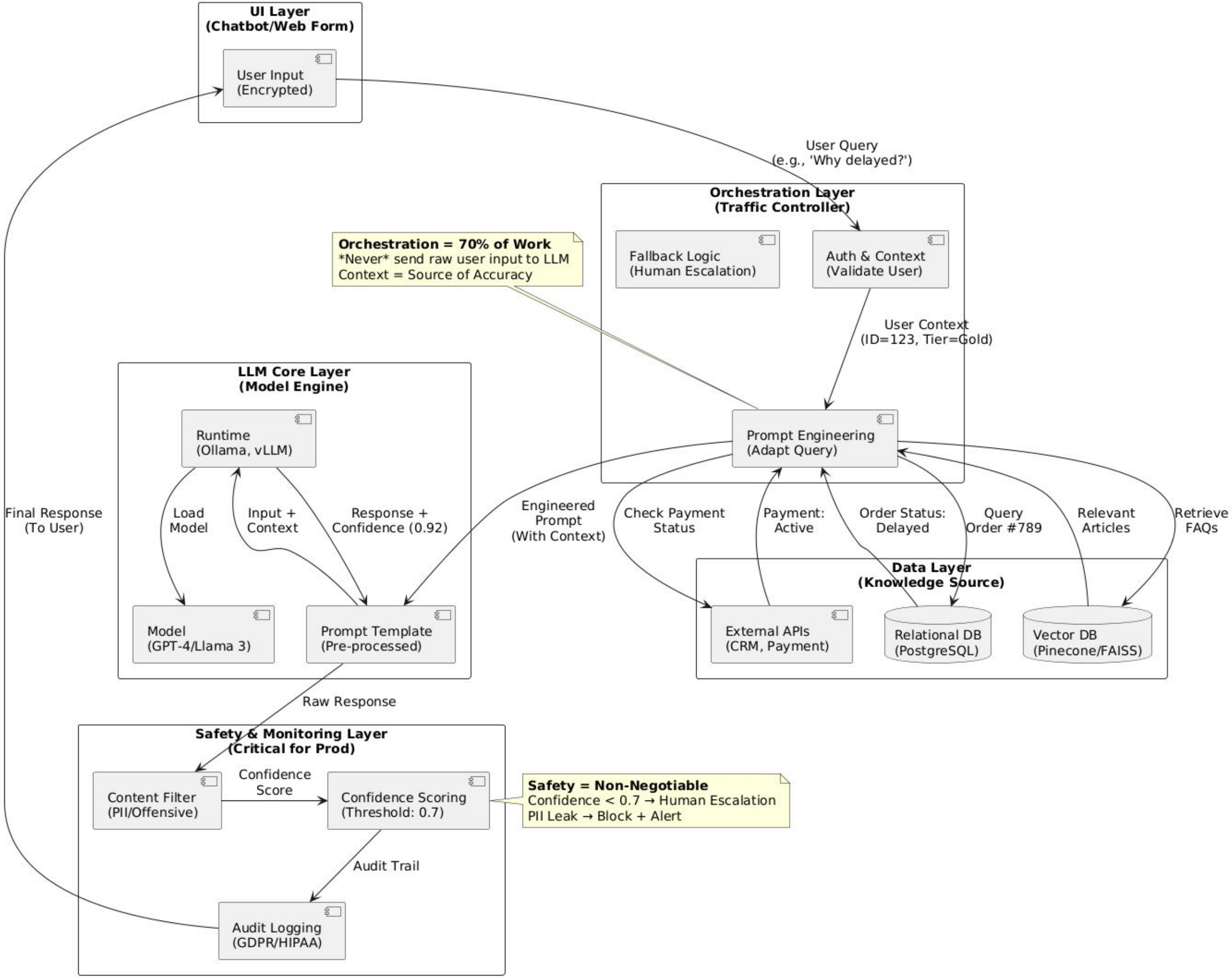
1. UI Layer
2. Orchestration Layer
3. Data Layer
4. LLM Core Layer
5. Safety & Monitoring Layer

High-level Architecture of an Agentic Application

Mastering Sof
Developme
A Guide For Early Career E
Balwinder Sodhi



Mastering Sof
Developme
A Guide For Early Career E
Balwinder Sodhi



UI Layer (Chatbot/Web Form)

User Input (Encrypted)

User Query (e.g., 'Why delayed?')

Orchestration Layer (Traffic Controller)

Fallback Logic (Human Escalation)

Auth & Context (Validate User)

User Context (ID=123, Tier=Gold)

Prompt Engineering (Adapt Query)

Orchestration = 70% of Work
Never send raw user input to LLM
Context = Source of Accuracy

LLM Core Layer (Model Engine)

Runtime (Ollama, vLLM)

Load Model

Input + Context

Response + Confidence (0.92)

Model (GPT-4/Llama 3)

Prompt Template (Pre-processed)

Engineered Prompt (With Context)

Check Payment Status

Payment: Active

Order Status: Delayed

Query Order #789

Relevant Articles

Retrieve FAQs

Data Layer (Knowledge Source)

External APIs (CRM, Payment)

Relational DB (PostgreSQL)

Vector DB (Pinecone/FAISS)

Raw Response

Safety & Monitoring Layer (Critical for Prod)

Content Filter (PII/Offensive)

Confidence Score

Confidence Scoring (Threshold: 0.7)

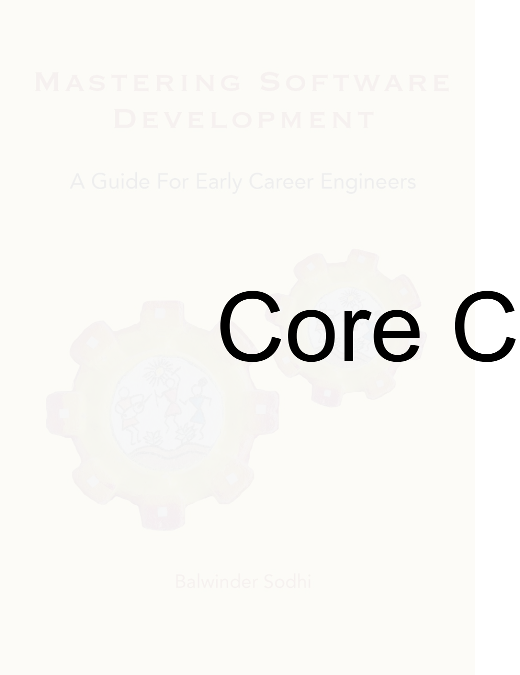
Safety = Non-Negotiable
Confidence < 0.7 → Human Escalation
PII Leak → Block + Alert

Audit Trail

Audit Logging (GDPR/HIPAA)

Final Response (To User)

Core Components and Interactions



MASTERING SOFTWARE DEVELOPMENT

A Guide For Early Career Engineers

Balwinder Sodhi

UI Layer

- Could be web, mobile, CLI, API endpoint
- Responsibilities:
    - Collect intents
    - Display results
    - Maintain interaction history
    - Provide grounding (context injection)
- Avoid embedding business logic here.

Mastering Software Development
A Guide For Early Career Engineers
Balwinder Sodhi

Orchestration Layer

- The heart of agentic systems.
- Responsibilities:
  - Loop management
  - State management
  - Selecting LLM prompts / strategies
  - Invoking tools
  - Handling fallbacks
  - Termination conditions
  - Logging, metrics, tracing
- This is where deterministic code constrains non-deterministic model behaviour.

Data Layer

Mastering Software Development
A Guide For Early Career Engineers
Balwinder Sodhi

- Contains:
  - Long-term memory (vector DB, relational DB)
  - Short-term scratchpads
  - Context stores
  - Knowledge bases
  - Cached computations
- Functions:
  - Retrieval
  - Storing agent state
  - Persisting workflow history
  - Guarding against hallucinations through retrieved ground truth

LLM Core Layer

Mainly includes:

- Model endpoints (OpenAI, Anthropic, local models)
- Prompt templates
- Sampling strategies (temperature, top-p)
- Tool-use capabilities
- System instructions
- Structured output handling (JSON schemas)

Safety and Monitoring Layer

Covers:

- Input/output guardrails
- Toxicity / jailbreak detection
- Role-based access controls for agent actions
- Observability: traces, metrics, logs
- Replay systems
- Circuit breakers for runaway loops
- Quotas & budget monitoring

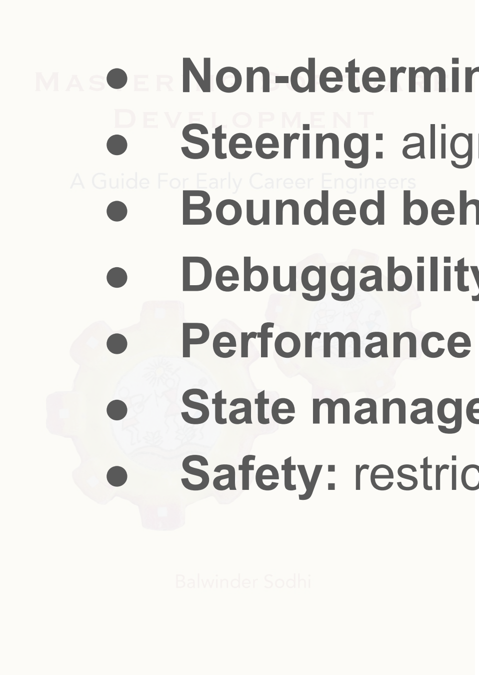
Architectural Foundations of LLM-Powered Software

MASTERING SOFTWARE DEVELOPMENT

A Guide For Early Career Engineers

Balwinder Sodhi

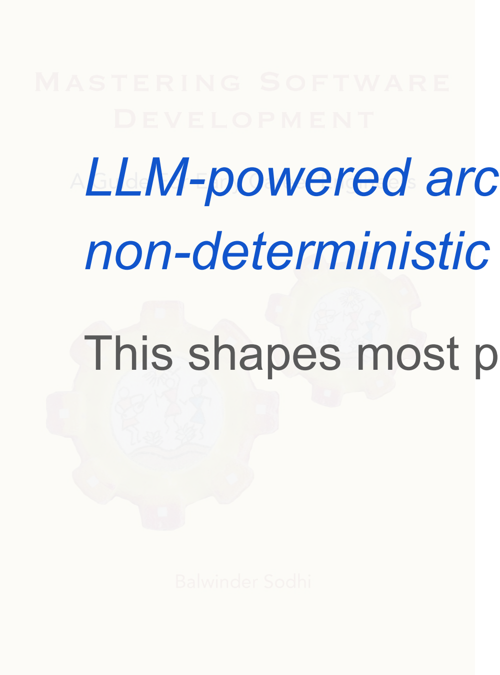
Key Architectural Challenges



Mastering Software Development
A Guide For Early Career Engineers
Balwinder Sodhi

- **Non-determinism:** LLM outputs vary
- **Steering:** aligning model output with business needs
- **Bounded behaviour:** preventing runaway loops or harmful actions
- **Debuggability:** difficult due to probabilistic reasoning
- **Performance optimization:** cost vs responsiveness
- **State management:** context growth, memory pruning
- **Safety:** restricting tool use and actions

Core Idea



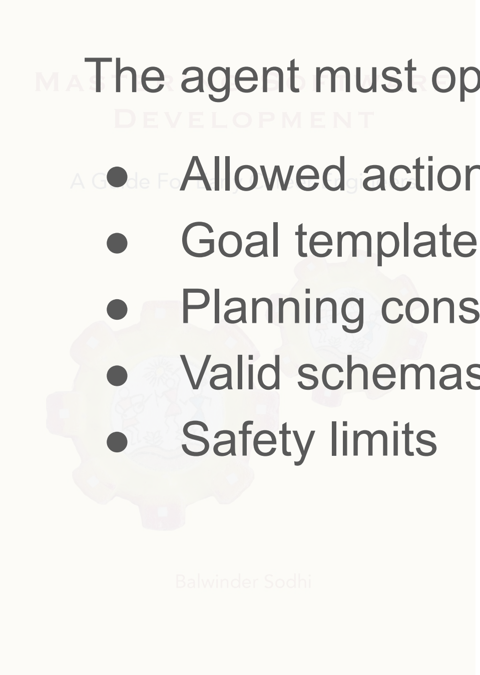
Mastering Software Development

Balwinder Sodhi

LLM-powered architectures must add deterministic structure around a non-deterministic core.

This shapes most patterns in modern AI systems.

The Agentic Architecture Pillars: #1 Structure



The agent must operate inside well-defined boundaries:

- Allowed actions
- Goal templates
- Planning constraints
- Valid schemas
- Safety limits

Pillar 2: Grounding

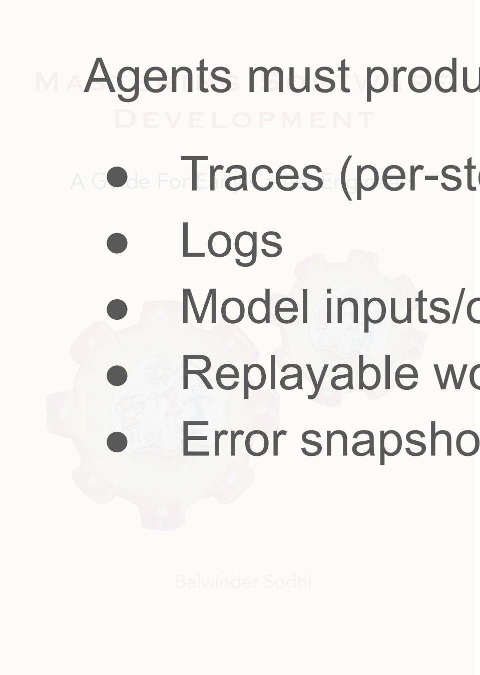
LLMs must operate on retrieved and validated context:

- Documents
- Databases
- Tool results
- Execution logs

Improves correctness and reduces hallucinations.

Mastering Software Development
A Guide For Early Career Engineers
Balwinder Sodhi

Pillar 3: Observability



Agents must produce:

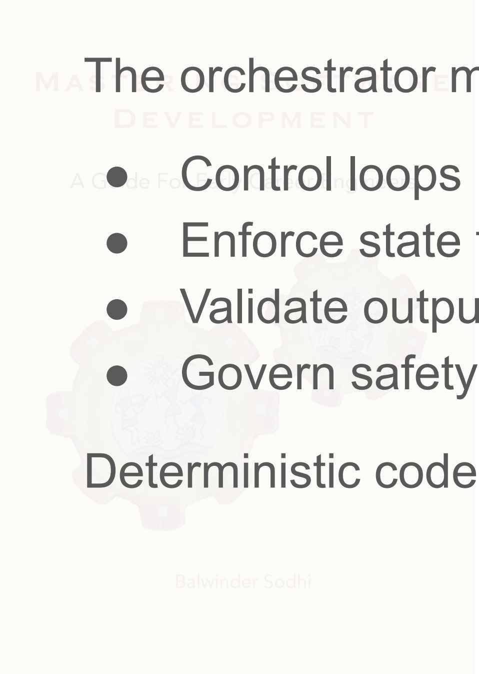
- Traces (per-step)
- Logs
- Model inputs/outputs
- Replayable workflows
- Error snapshots

Pillar 4: Deterministic Control

The orchestrator must:

- Control loops
- Enforce state transitions
- Validate output
- Govern safety and tool usage

Deterministic code contains the model.



Mastering Software Development
A Guide For Early Career Engineers
Balwinder Sodhi

Architectural Patterns: Making Trade-offs Explicit

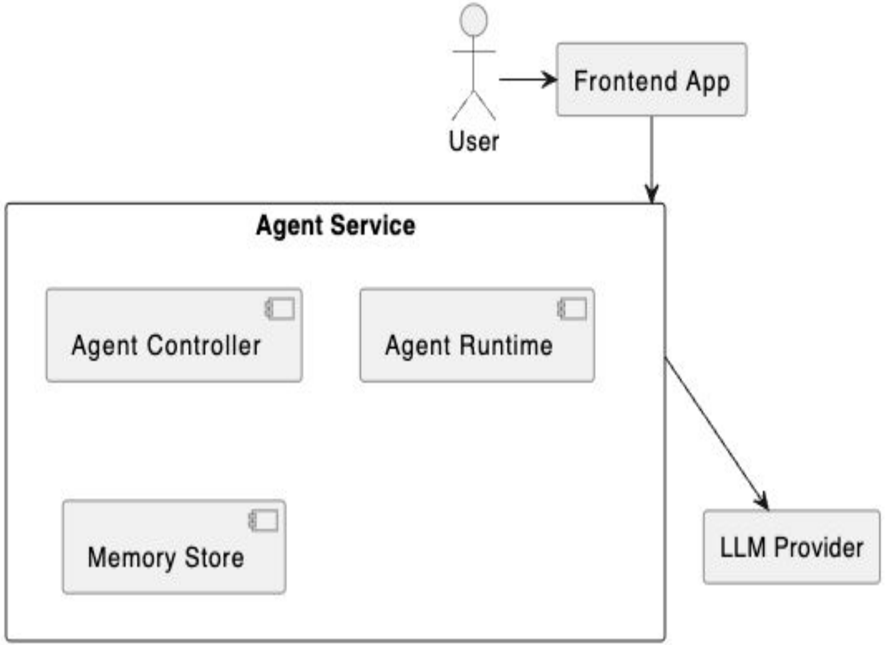
Agent as a First-Class Service

Mastering Software Development
A Guide for Early Career Engineers
Balwinder Sodhi

Key idea

Treat the agent like any other backend service:

- Has an API
- Maintains state
- Exposes capabilities
- Instances can be scaled horizontally
- Decouples agent execution from the UI



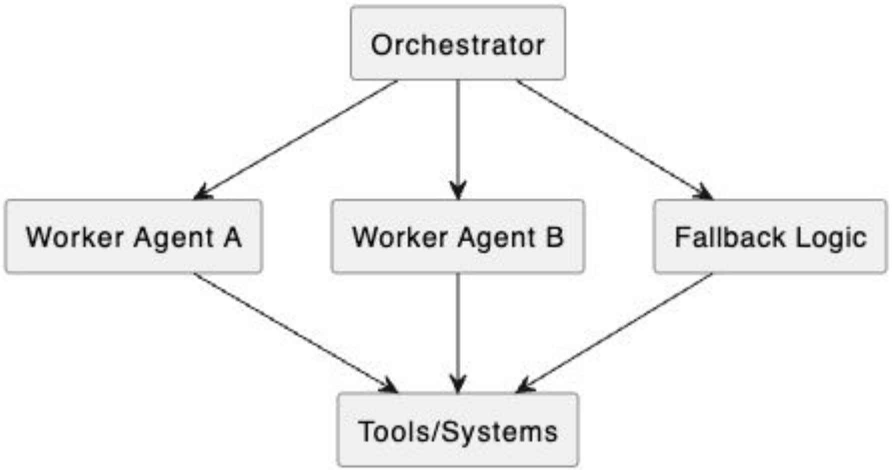
Orchestrator–Worker With Explicit Fallbacks

Mastering Software Development
A Guide For Early Career Engineers
Balwinder Sodhi

Key idea: Use an orchestrator agent that delegates tasks to worker agents or tools, with fallback logic baked in.

Benefits:

- Robustness under uncertainty
- Clear responsibility boundaries
- Modular task decomposition

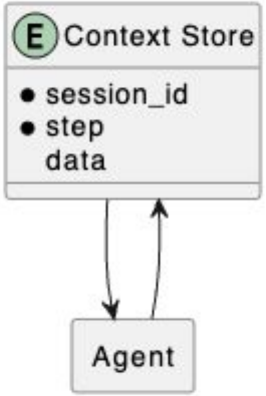


Context as a First-Class Data Type

Key idea: Context must be explicitly managed:

- Passed to every agent step
- Versioned
- Composable
- Persisted for replay
- Stored in structured formats

Prevents “lost context” errors and uncontrollable behaviour.

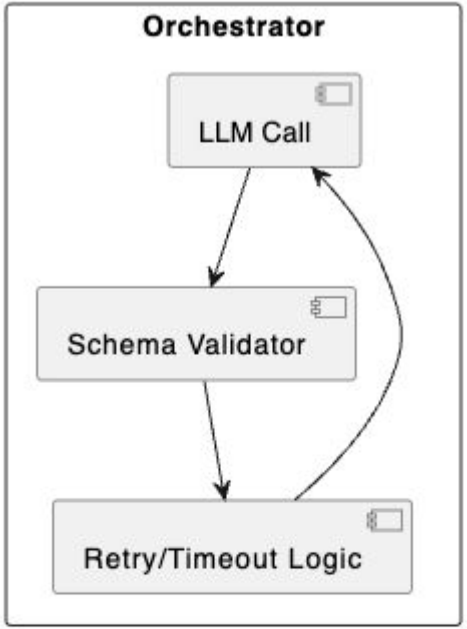


Deterministic Boundaries for Non-Deterministic Systems

All LLM outputs are validated:

- JSON schema enforcement
- Enum constraints
- Tool selection constraints
- Step timeouts
- Retry policies
- Max depth, max steps

Deterministic boundaries create safe operational envelopes.

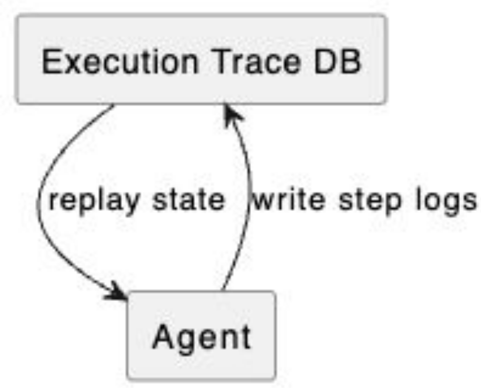


MASTERING SOFTWARE DEVELOPMENT
A Guide For Early Career Engineers
Balwinder Sodhi

Observability and Replayability

Mastering Software Development
A Guide For Early Career Engineers
Balwinder Sodhi

- Stores:
  - Prompts
  - Inputs
  - Outputs
  - Tool invocation logs
  - Execution traces
- Used for:
  - Debugging
  - Governance
  - Safety audits
  - Improvement cycles



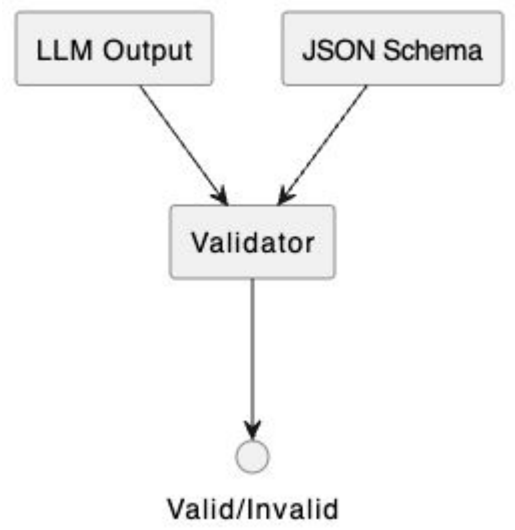
Schema-Governed Contracts

Mastering Software Development
A Guide For Early Career Engineers
Balwinder Sodhi

LLM output must conform to explicit schemas:

- JSON schema
- DSLs
- Strict enums
- Bounded numeric ranges
- Plan schema (steps, actions, dependencies)

Helps constrain non-deterministic text generation.



Key Takeaways

- Not all LLM usage requires agents
- Agentic apps introduce complexity—use them wisely
- Architecture must constrain LLM unpredictability
- Orchestration is where most engineering value lies
- Observability and safety are essential
- Context and schemas reduce hallucinations
- Patterns help structure systems under uncertainty