# Agentic AI Architecture

A Deep Dive into Autonomous Systems, Design Patterns, and Implementation Frameworks

# Agenda

* Agentic AI Architecture
* Agentic Architecture Types
* Key Components of the Agentic AI Framework
    * Perception Module
    * Cognitive Module
    * Action Module
    * Learning Module
    * Collaboration Module
    * Security Module
* Agentic AI Design Patterns
    * Reflection Pattern
    * Tool Use Pattern
    * Planning Pattern
    * ReAct (Reasoning and Acting)
    * ReWOO (Reasoning WithOut Observation)
    * Multi Agent Pattern
* Design Considerations

# Core Concepts

Understanding the paradigm shift from Automation to Autonomy

# The Evolution of AI Systems

## Traditional Automation

Systems follow rigid, pre-defined rules. If condition A is met, execute action B. These systems are deterministic, fragile to change, and require manual intervention for edge cases. They "do" exactly what they are told.

## Agentic AI

Systems possess agency. They are given a high-level goal and must autonomously determine the steps to achieve it. They perceive their environment, reason about actions, use tools, and adapt to feedback to solve complex problems.

# The Agentic Control Loop

## Perceive, Reason, Act

At the heart of every agent is a continuous control loop.

- The agent observes the state of the world (Perception),
- Processes this information against its goals and memory (Cognitive),
- Executes commands via tools (Action).
- This loop enables the system to interact dynamically with its environment, creating a feedback mechanism that allows for error correction and multi-step problem solving.

# Agentic Architecture Types

## Single Agent

A solitary autonomous entity with a unified memory and toolset. Best for linear, straightforward tasks where context switching is minimal and global context is required.

## Multi-Agent System (MAS)

A network of specialized agents working together. Each agent focuses on a distinct domain (e.g., Coder, Reviewer). Best for complex, multifaceted problems requiring diverse expertise.

# Key Components

Deconstructing the Agentic Framework

# 1. Perception Module

## Sensing the Environment

The perception module is the agent's interface with the world. It converts external signals into a format the Cognitive Module can process.

- **Multimodal Inputs:** Text, Images, Audio, and Video.
- **Digital Signals:** API responses, database states, and system logs.
- **User Intent:** parsing complex natural language instructions.

# 2. Cognitive Module

## The Brain (LLM)

This is the core processing unit, typically a Large Language Model (LLM). It handles reasoning, planning, and decision-making.

It maintains **Memory** (Short-term context and Long-term vector stores) to retain conversation history and knowledge. It uses this context to decide *which* tool to use and *how* to use it.

# 3. Action & 4. Learning Modules

## Action Module

The "hands" of the agent. This module executes the decisions made by the brain. It includes:

- Tool Use: Calling external APIs (search, calculator)
- Effectors: Manipulating files, sending emails, or controlling robotic hardware.
- Output: Generating final responses

.

## Learning Module

How the agent improves over time. Unlike static scripts, agents can adapt.

- In-Context Learning: Adapting behavior based on prompt history.
- RAG (Retrieval): Fetching relevant knowledge to update its context window.
- Fine-tuning: Updating model weights for specific tasks.

# 5. Collaboration & 6. Security

## Collaboration Module

Essential for Multi-Agent Systems. It defines how agents talk to each other.

- **Orchestration**: A central manager delegates tasks.
- **Choreography**: Agents hand off tasks autonomously.
- **Shared Memory**: A blackboard for shared state.

## Security Module

The guardrails preventing rogue actions.

- **Input Validation**: Preventing prompt injection.
- **Action Permissibility**: Limiting API scope (e.g., read-only access).
- **Human-in-the-Loop**: Requiring approval for high-stakes actions.

# Design Patterns

Architectural blueprints for robust agents

# Pattern: Reflection

## Self-Correction Loop

Reflection allows an agent to critique its own output before finalizing it. This mimics human revision.

- 🔄 **Workflow:** Generate -> Critique -> Regenerate.
- ✓ **Benefit:** Significantly reduces hallucination and logical errors.
- </> **Use Case:** Code generation (writing code, running tests, fixing errors based on stderr).

# Patterns: Tool Use & Planning

## Tool Use

Extending the LLM's capability beyond text. The model outputs structured JSON to call functions (e.g., `get_weather(location)`), bridging the gap between language and logic.

## Planning

Decomposing complex goals. Techniques like **Chain of Thought (CoT)** and **Tree of Thoughts (ToT)** break big problems into manageable, sequential steps.

# Reasoning Architectures

## ReAct

Reason + Act. The agent reasons about the state, takes an action, observes the output, and repeats.

*Pros:* Highly accurate, handles dynamic environments well.
*Cons:* High latency, sequential execution creates bottlenecks.

## ReWOO

Reason WithOut Observation. The agent generates a full plan *upfront* with placeholders for tool outputs, then executes tools in parallel.

*Pros:* Much faster, lower token cost.
*Cons:* Less adaptive if a tool fails unexpectedly.

# Multi-Agent Patterns

## From Solo to Squad

Complex problems often exceed the context window or reasoning capabilities of a single model. Multi-Agent Systems (MAS) solve this by distributing tasks.

- **Specialization:** Agents assume distinct personas (e.g., 'Researcher', 'Writer’).
- **Orchestration:** A root agent manages the workflow and consolidates results.
- **Scalability:** Parallel execution reduces total processing time.

# Multi-Agent Collaboration Models

- **Hierarchical (Boss/Worker):** A "Manager" agent breaks down the task and assigns sub-tasks to specialized "Worker" agents (e.g., Coder, Researcher). The Manager aggregates results.
- **Sequential Handoffs:** A linear chain where the output of Agent A becomes the input for Agent B. Ideal for well-defined pipelines (e.g., Write -> Review -> Publish).
- **Joint Collaboration:** Agents act as peers in a chat room, discussing the problem until a consensus is reached. Good for creative brainstorming.

# Design Considerations

Challenges in
Production

# Implementation Challenges

## Latency

### Inference Time

Chained calls add up. Parallelization (ReWOO) is key.

## Cost

### Token Usage

Verbose reasoning loops can be expensive. Optimize prompts.

## Loops

### Error Recovery

Agents can get stuck in infinite retry loops. Set max-steps.